Emerging Computing Technologies in HEP

Amir Farbin
University of Texas at Arlington

Overview

- Meant to be an introduction to get you thinking
  - I'll be speculative... not very detail.
- HEP Computing Challenges
- Emerging Technologies
- One slide on Virtualization... one on Cloud Computing.
- Solid State Drives
- General Processing Graphics Processing Units

HEP Computing Problems

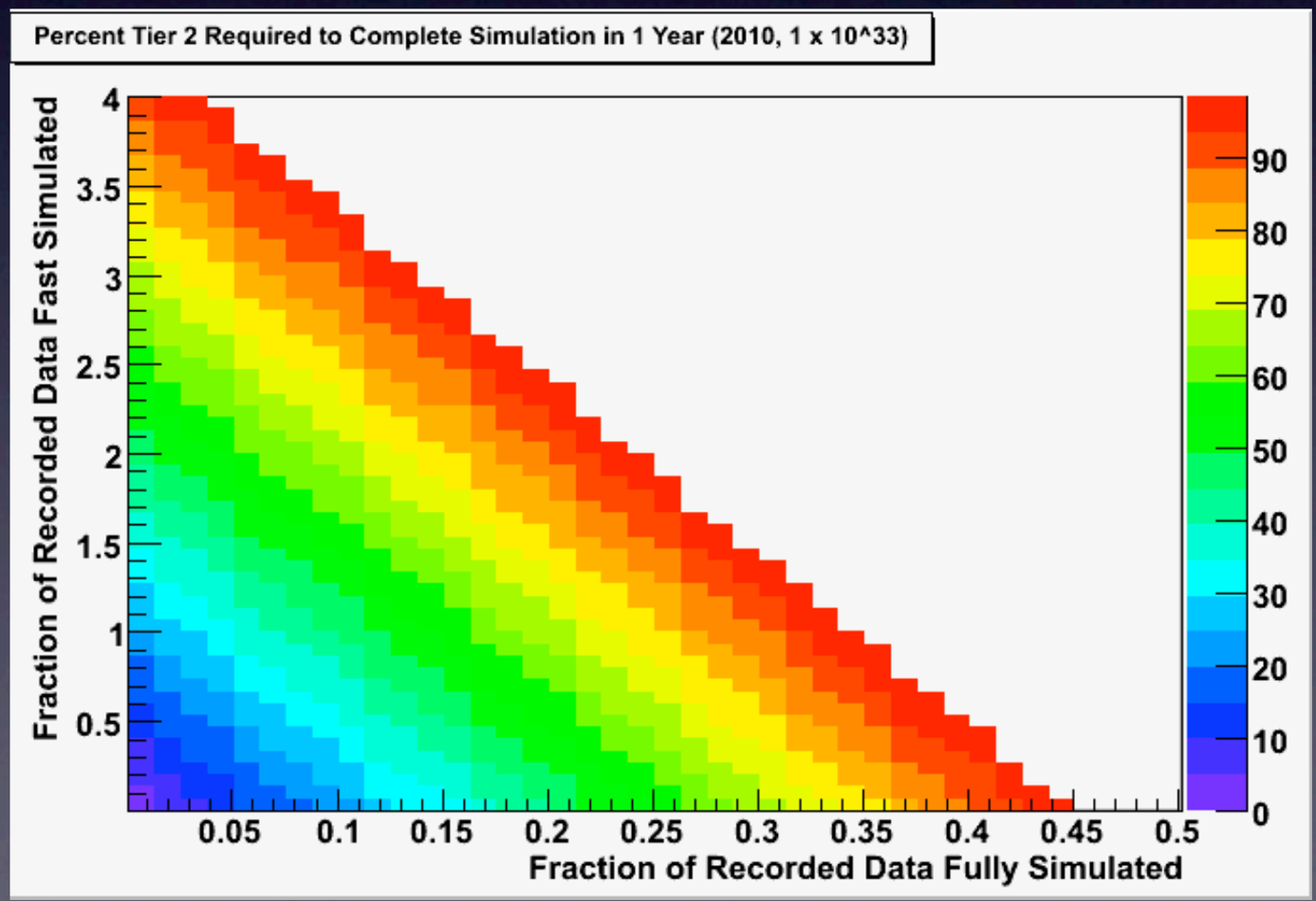
1. *Moore's Law*: transistors double every 18 months (true)... but HEP (and almost everyone else) has implicitly assumed that it is clock speed that doubles (false).

    - Chips have become smarter with more transistors...
        - eg circuitry which automatically "parallelize" instructions... or better/more cache.
        - Limited... CPUs can be just so smart given code which isn't fundamentally parallelizable.
    - Chips ran too hot at high clock speeds so the current trend is Multi-core processors rather than faster processing cores.
        - Naive Interpretation: Since most HEP applications are embarrassingly parallel (each event is independent) then N times more cores = N times the processing rate.
            - Not true because: HEP applications use a lot of memory... so you need N times cores + N times memory to get N times the processing rate.
    - Next generation operating systems (eg Snow Leopard) are being rewritten in response to the shift to multi-core.

Computing Problems in HEP

2. Detailed Simulation (ie Geant) of lots of particles (ie Jets) through complex detector geometries (ie LAr) takes lots of processing. (2000-3000 kSI2K s ... ie 10s of mins)

- Clearly can't do analysis with MC sample == 10% of collected data.
  - Must use fast simulation.
- Consider: Percentage of Tier 2 CPU required for simulation production as function of fraction of 2010 recorded data which is fully and fast simulated.
- Need at least 80% of tier 2s for MC production to get 10% full/ 300% fast of a year's worth of data.
- Leaves only 20% for analysis on tier 2!



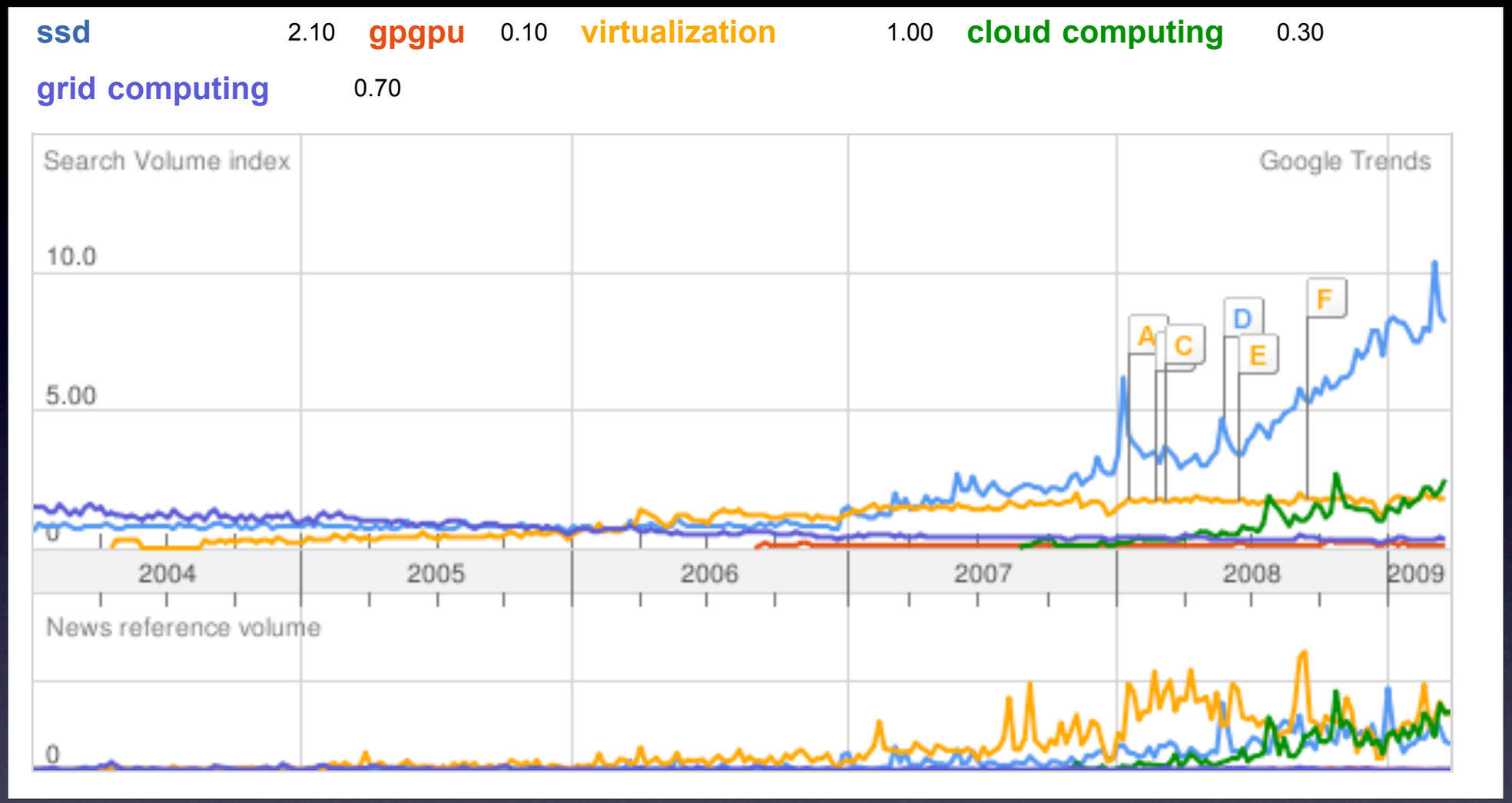
Computing Problems in HEP

3. Computing Models typically do not include resources for the CPU-intensive analysis activities which in the past decade have come to characterize analyses of HEP data at the Tevatron and the B factories:
   - sophisticated fits, statistical analysis of large "toy" Monte Carlo models, matrix element calculations, and use of the latest discriminant techniques, such as boosted decision trees.
   - Huge requirements before conferences...
4. Setup and maintenance of highly distributed systems (like GRID) requires local attention and expertise.

Solutions?

- Use modern processors more efficiently... means Parallelization
  - Today it mostly means running more processes, but using shared memory. eg magnetic field/detector geometry/algorithms can be shared jobs.
  - Parallelize your HEP code... difficult because it is complicated and we didn't design our software for parallel computing.
  - Parallelize specific slow steps... (More on this later).
- Access to more computing
  - Take simulation load off of tier 2... letting more analysis at tier 2 which has all of the data (AOD/DPD).
    - Use Tier 3s
    - Use leveraged resources (eg opportunistic computing).
- Simplify deployment

Emerging Technologies



- While GRID computing is becoming less popular, other major trends are emerging:
  - *Virtualization*- Emulate virtual machine(s) on physical machines.
  - *GPGPUs* (General Purpose computing on Graphics Processing Units)- most of this talk
  - *Solid State Drives* (SSDs)- No moving parts = fast read/write and random access (most important for HEP).
  - *Cloud Computing*- Leasing resources from companies as needed.

Virtualization

- I'll focus on GPGPU & SSDs... a few comments on VMs and CC.
- Virtualization is fairly common place now.
  - *Packaging*: Most common example in HEP. CernVM allows running experiment software on any platform (eg laptop). Others: packaging of difficult to deploy GRID services for Tier 3s.
  - *Opportunistic Computing:* Most promising but unrealized: Idle CPUs pick up jobs from GRID... heavy payloads make this technically difficult. Working example at University of Oklahoma using CoLinux (technically no a VM) & D0 software.
  - *Site Management:* Examples out there... but not absolutely clear if VMs simplify or complicated things.
- Virtualization is perhaps no longer "emerging"... but not all hopes yet realized.

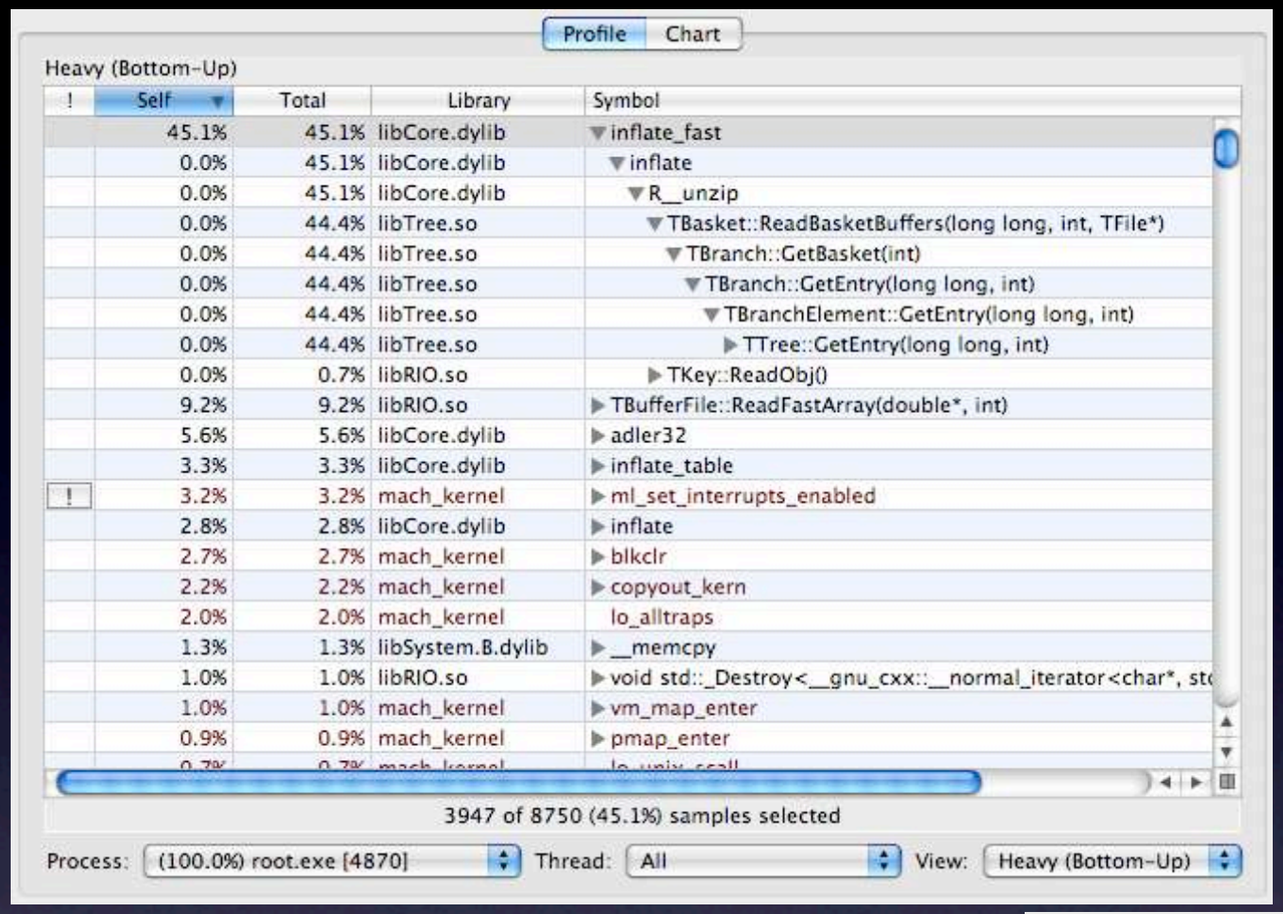
Cloud Computing

- Definition not clear... basically on-demand of access to resources (CPU/ Storage).
    - Both in terms of location and cost.
- Implicit reliance on Virtualization...
- Nimbus is notable HEP implementation. Turn any resource (eg Amazon EC2) into a self-configuring cluster.
- General Problems:
    - Reliability/Performance not spectacular
    - Cost can be prohibitive.
    - Proprietary standards. Industry (eg Amazon) leaders have vested interested in keeping it so.
        - IBM + other companies introduced the "Open Cloud Manifesto"... acceptance
- Lots of buzz about the "Myth of Cloud Computing"

Solid State Drives

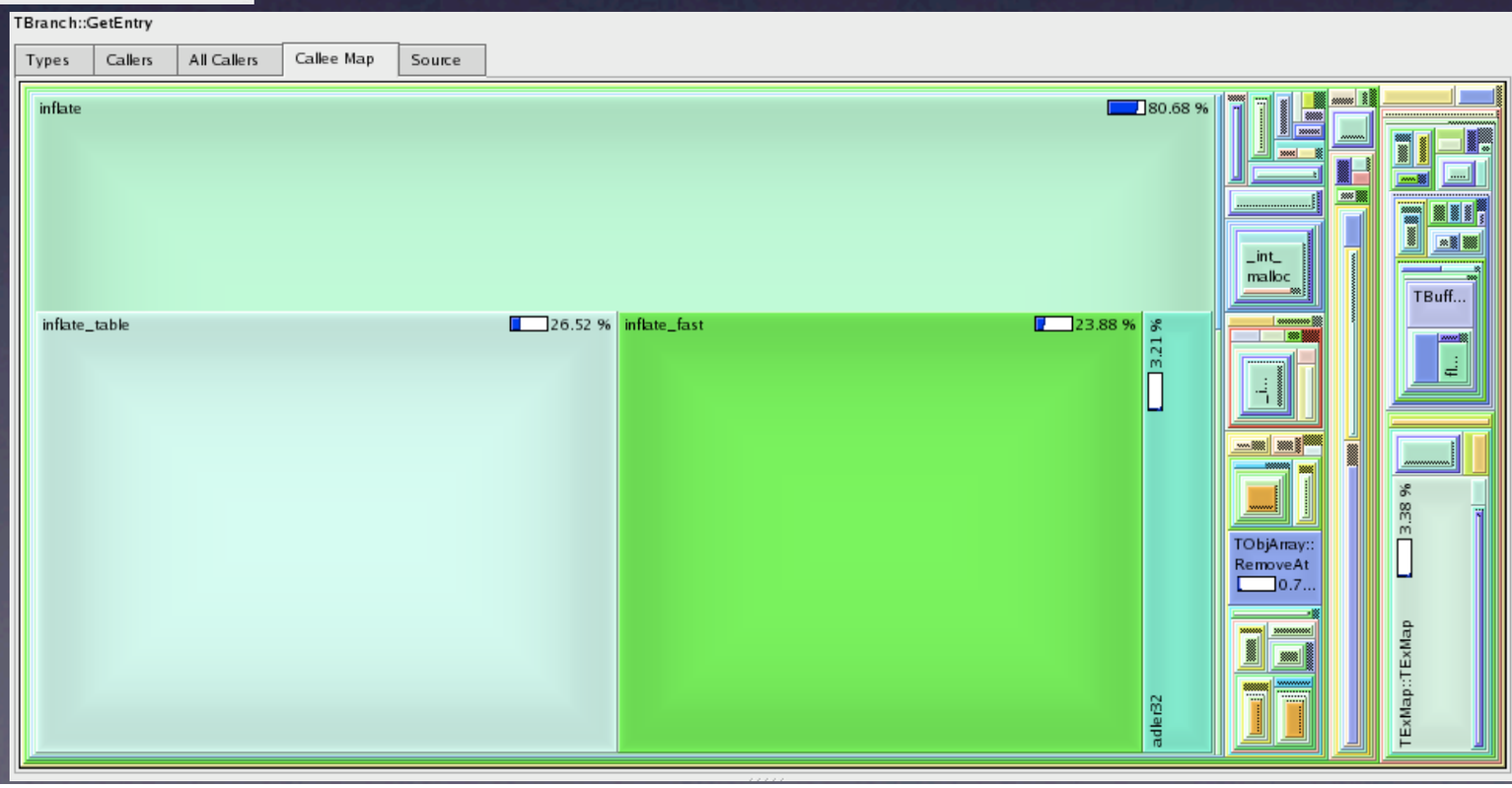
- No moving parts... fast read/write... especially random access.
- Sequential rates of ~250MB/s on laptop drives to ~1.5GB/s on specialized cards (eg ioFusion).
- Rapidly dropping prices.
- Relevance to HEP? Applications (eg simulation) are generally CPU bound, but:
  - Analysis can be IO bound... especially on systems where 100's of jobs rapidly read data.
  - Data transfer (copy).

ROOT I/O



- The limiting factor in ROOT I/O is not disk speed... it's decompression (CPU limited).
- > 60% time reading Simple Data (ntuples) is spend in decompression.

- Majority of function calls are in decompression algorithms.



Read/Write Tests

- Simple "flat" ntuples are the simplest, therefore fastest, format...
- Controllable test:
  - *Write Application*: Create ntuples with simple types (bool,int,float) and vectors of simple types... random values/lengths
  - *Read Application*: Histogram every quantity in ntuple.
  - Results stabilize with 20 quantities of each type (3 KB/event), and > ~600 events.
- Should we compress?
  - ROOT "gzip" compression: 0-9 Levels
  - 500k events = 1.5 GB uncompressed. 1.1-1.2 GB with compression.
  - Lowest Compression Level = ~5x reduction in write rate!
    - Maybe we should evaluate if we should spend more on disk or CPU for endpoint analysis.

Hard vs Solid State Drive

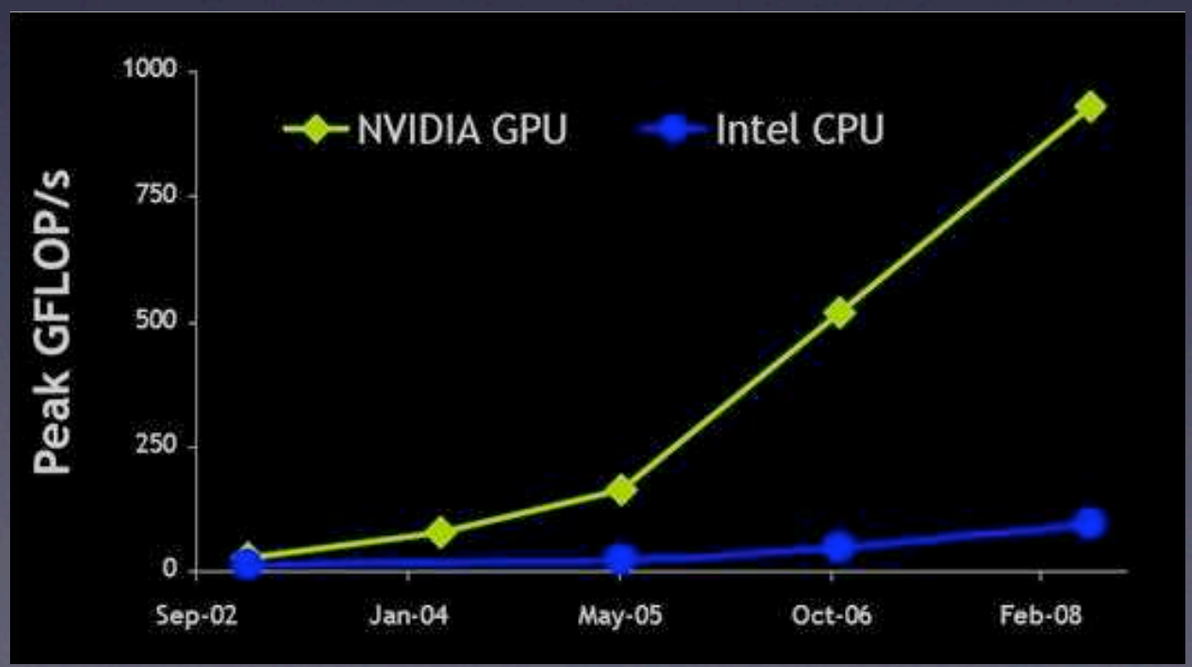
- Sequential Read (Write): ~70 (70) MB/s vs 250 (180) MB/s...
- Random depends on data size and pattern.

CPU Limited / Disk Limited	Simultaneous Jobs	Compression Level	Rate (MB/s)	
			Hard Drive	Solid State Drive
Writing	1	0	25	25
	8	0	50	75
	1	9	4	4
	8	9	25	25
Reading	1	0	24	32
	8	0	20	28
	1	9	16	20
	8	9	14	17

- Generally CPU limited. ~ 5x write speed reduction with compression.
- With 8x simultaneous access, only uncompressed write is clearly disk limited.
- SSDs help with reading in all cases (improved random access?)
- 2x speed improvement w/ no compression + SSD w/ 8 simultaneous jobs.

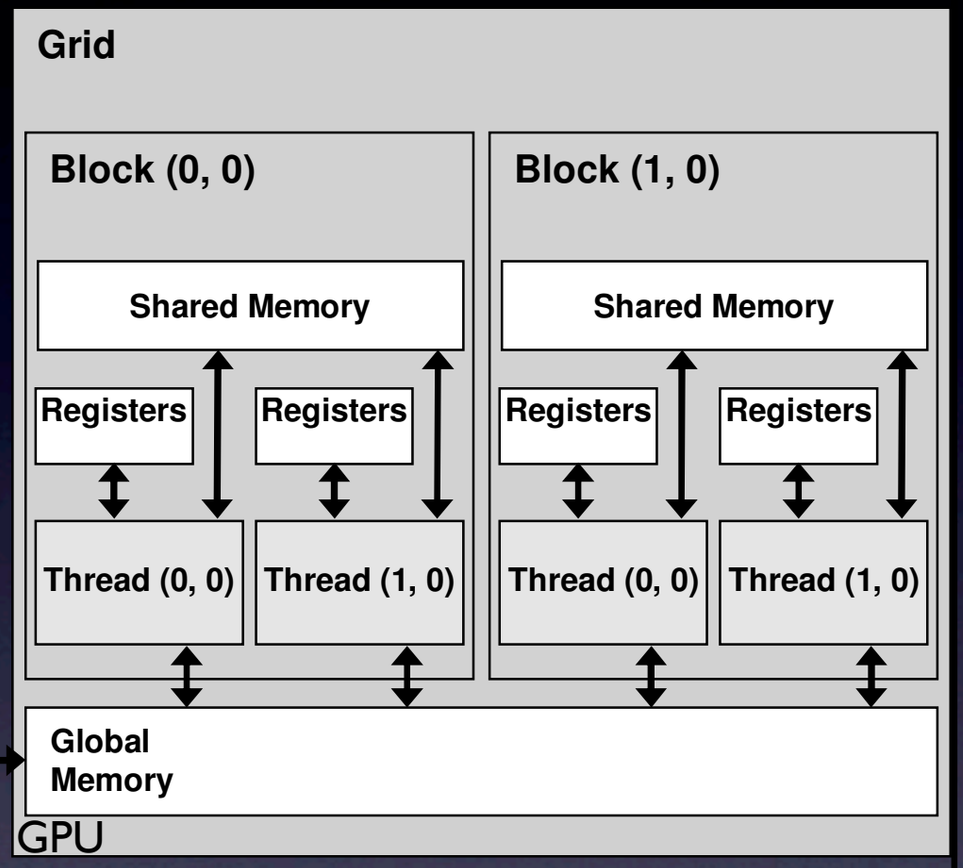
What is GPGPU?

- General Purpose computing on Graphics Processing Units.
- Historically GPU development driven by gaming industry.
  - Specialized co-processors + video memory which simultaneously perform the same operation (generally linear algebra) on multiple chunks of data.
    - Single Instruction Multiple Data (SIMD) computing.
  - Originally GPUs had lots of simple processing units capable of simple instructions.
  - Evolved into execute nearly the same instructions as standard CPUs.
    - GPUs today have 100's of cores, and are capable of simultaneously running O (10000) threads... achieving 100's of GFlops on a single GPU.
- CPUs are optimized for low latency... GPUs are optimized for high throughput.
- These GPUs are already everywhere!
- Some new Macs have 2 GPUs!



GPGPU Computing

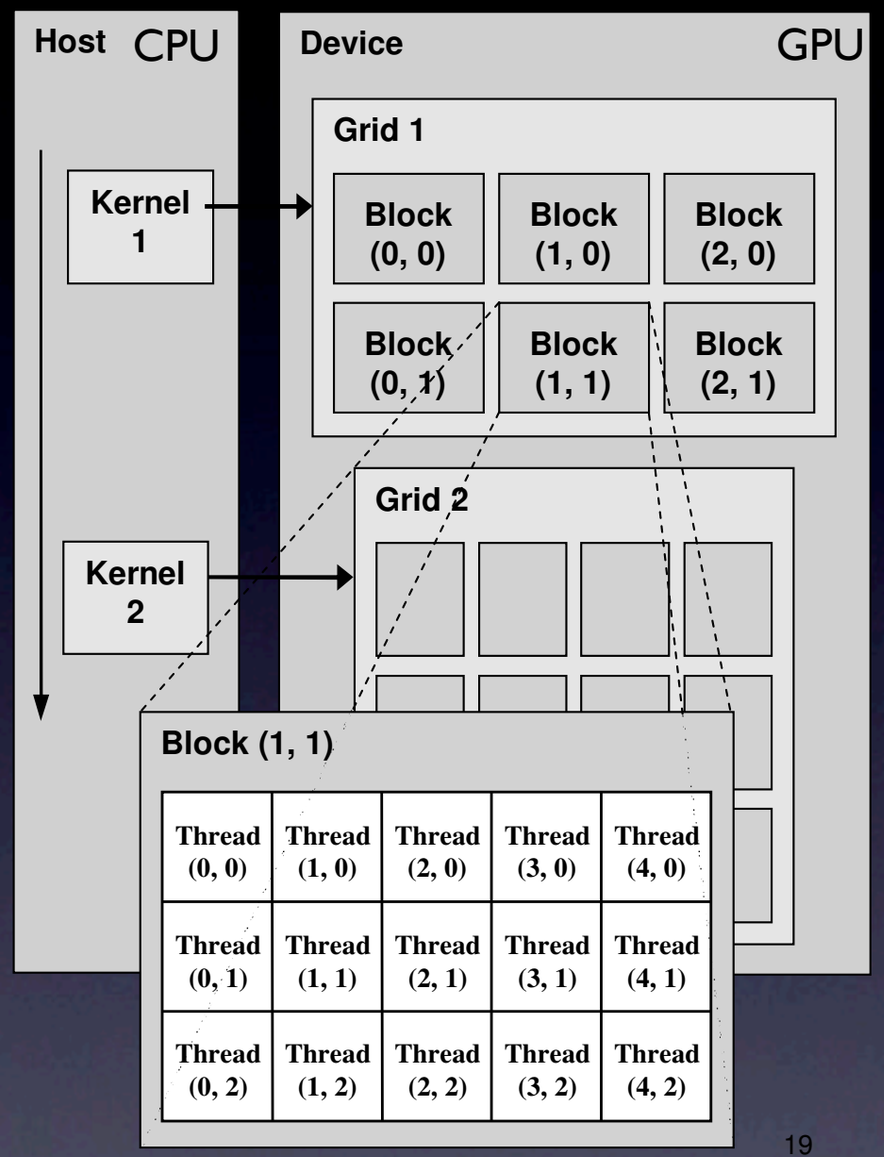
- An individual core in a GPU is generally not as powerful or fast as a CPU.
- GPU is optimized for parallel computing:
  - It's all about data parallelism.
  - Memory access can be costly. Different types of memory provide different latency.
  - A processing unit with a thread waiting for data switches to another thread. (All in Hardware).
  - Better performance when treads perform same operation (in step).



- Computation Model
  - Data is moved (possibly asynchronously) between CPU and GPU memory.
  - Kernel is a computation initiated by CPU to run on GPU.
  - Data is broken into blocks, which is assigned to a physical unit... Multiple treads process each block, with low latency access to data in that block.
    - Allows transparent scaling to different GPUs.

GPGPU Computing

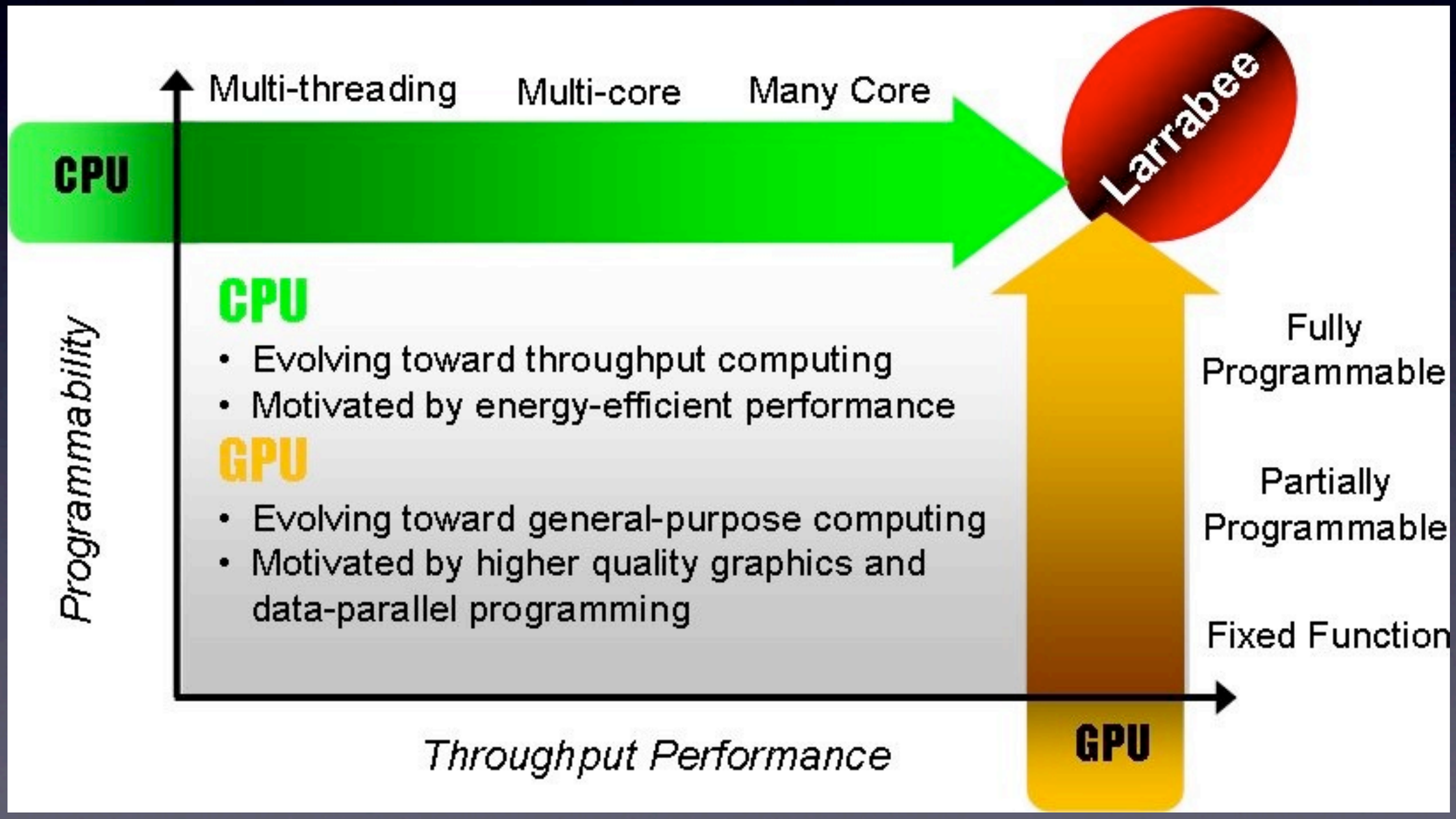
- An individual core in a GPU is generally not as powerful or fast as a CPU.
- GPU is optimized for parallel computing:
  - It's all about data parallelism.
  - Memory access can be costly. Different types of memory provide different latency.
  - A processing unit with a thread waiting for data switches to another thread. (All in Hardware).
  - Better performance when treads perform same operation (in step).
- Computation Model
  - Data is moved (possibly asynchronously) between CPU and GPU memory.
  - Kernel is a computation initiated by CPU to run on GPU.
  - Data is broken into blocks, which is assigned to a physical unit... Multiple treads process each block, with low latency access to data in that block.
    - Allows transparent scaling to different GPUs.



19

CPU/GPU Convergence

- Intel is also working the other way...Larrabee will be lots of simple x86 CPUs on one chip.



GPGPU Application Development

- Ideal GPGPU Applications exhibit following characteristics:
  - *Compute Intensity* – Large number of arithmetic operations per IO or global memory reference.
  - *Data Parallelism* – this exists in a dataset if the same function is applied to all records of an input stream and a number of records can be processed simultaneously without waiting for results from previous records.
  - *Data Locality* – a specific type of temporal locality common, where data is produced once, read once or twice later in the application, and never read again.
- In general, code developed for CPU must be explicitly parallelized to run on GPU.
  - Must consider how to break data into chucks, with simple algorithm processing each chunk.
  - Model: data is prepared on CPU, shipped to GPU, processed, returned.
- Development environments generally provide extensions to C which allow integration of CPU/GPU code.

Hardware/Software Landscape

- *Hardware Vendors*: NVidia, ATI (AMD now), ClearSpeed, ...
- *Software Development Environments*: CUDA (NVidia only), BrookGPU, AMD Stream Computing (AMD only), Sh, OpenCL
- In general, NVidia has been pressing GPGPU the most.
  - O(100) of scientific and commercial applications already developed in CUDA.
  - Lots of examples on NVidia website.
  - Many commonly used libraries already have CUDA implementations (eg LAPACK).
- Most promising development environment is OpenCL (Open Computing Language).
  - Initiated by Apple, passed to open consortium, endorsed by nearly everyone.
  - Architecture independent parallel computing for CPU and GPUs... use all resources!
  - Specs release late 2008. 1st implementation will be in Snow Leopard (next Mac OS X) later this year.
  - AMD has shown OpenCL example using their ATI GPUs.
  - Sufficient similarity to CUDA... develop in CUDA now... migrate to OpenCL later.

HEP Applications (I)

- Many applications in ROOT:
  - *ROOT I/O*: Data decompression is the largest bottleneck to high speed analysis.
  - *Fitting and Minimization*: RooFit, Minuit2 (OO Minuit), etc.
    - Likelihood calculation is highly factorizable (over samples, events, hypothesis, PDFs,...)
    - Much time is spent in PDF normalization via MC integration... very parallelizable.
  - *Statistical libraries*: TMVA, MLP, New Stat Tool.
    - eg: Evaluation of Boosted Decision Trees is very time consuming, but lends itself to parallelization.
  - *Linear Algebra*: TMatrix, SMatrix.
  - *MathCore*: Physics Vectors Basic algorithms, Math functions, etc.
  - *MathMore*: Random Numbers, Extra algorithms, Extra Math functions.

HEP Applications (II)

- *Generators*- Speed up matrix element calculations.
  - Lots of time spent in integration.
  - Also possibly parallelizable at event level.
- Detector *Reconstruction* Algorithms:
  - 60x track fitting boost shown Al-Turany *et al* (CHEP 2009).
  - Byte-stream decoding (a big bottle-neck), track fitting, clustering, jet-finding, etc.
  - Huge implications for High-Level Trigger.
  - Use GPGPUs in Read-Out-Drivers? (Much easier to program than DSPs/FPGAs)
- Detector Simulation (Geant 4).
  - Holy Grail... but very difficult. May require complete rewrite.
  - Event-level vs Particle-level parallelization?
  - Access to geometry/magnetic field difficult.

First Steps

- Detailed evaluation of ROOT decompression.
  - gzip compression is not parallelizable as is...
    - worked out format modifications that would make it possible.
      - Need block boundaries. 1-2% additional info.
- Evaluation of CUDA and OpenCL (developer release).

Simple GPU Test

- Monte Carlo integration of n-dim function. Important in:
  - Generators...
  - Matrix Element Methods.
  - Maximum-likelihood fits... often limits complexity of models.
- First implementations.
  - Mostly a learning experience... easy to do things wrong and get non-optimal results.
  - Problems with random numbers... in principle resolvable.
  - Find only ~ 3x speed improvements on previous generation GPUs over CPU.
    - Lots of phase-space for optimization.

GPU Programming Experience

- *OpenCL*: Compiles GPU code for current architecture at run-time while setting up the calculation.
  - Strange bugs... compiler seems not mature.
- *CUDA*: Single Architecture allows compiling at compile time.
  - Faster execution than OpenCL.
- Code running on GPU must be very simple...
  - *Data*: May only use simple types + arrays of simple types (and structs)...
  - Only call functions with simple types. No passing of pointers, so no array passing!
    - Makes even the MC integration example hard to generalize.
    - Must think of appropriate programming model.
- Data transfer between CPU/GPU memory is time consuming.
- Performance highly dependent on how data is organized... high price for poorly accessing memory.

Migrating from CPU to GPU

- You can't just recompile your code.
  - In general, requires rethinking, redesign, and lots of optimization.
- Model: Use GPU as a parallel co-processor...
  - most likely scenarios is to use GPU to assist with specific time consuming steps.
- Geant4 "Fantasy":
  - Run lots of parallel Geant4 threads.
  - Each talks to "Magnetic Field Service" which batch processes extrapolations on GPU.

Final Comments

- If your analysis is IO limited, consider decompressing your ntuples to get faster analysis on your Tier 3s,.
- Adapting to new technologies is not so simple (SSDs excepted)
- GPGPU is promising...
  - But very challenging... requires R&D.
- In order to take advantage of future processors, next generation HEP software should be designed for parallelization.

In the news

- Last week NVidia + HP provided first GPU + Server capable of providing GPU to virtual machine.
  - GPU has virtualization extensions (like the new Intel CPUs) which is supported by the Server.
  - VMWare will support the GPU extensions.
- IBM + other companies introduced the "Open Cloud Manifesto".
  - Make services open rather than proprietary so cloud services can be deployed by anyone and clients can migrate.
  - Unsurprisingly big players in Cloud Computing didn't sign (eg Amazon, Microsoft) so they don't loose their edge!
- FusionIO will soon release 1.3 TB SSDs cards with 1.5 GB/s read and <50 µsec latency.