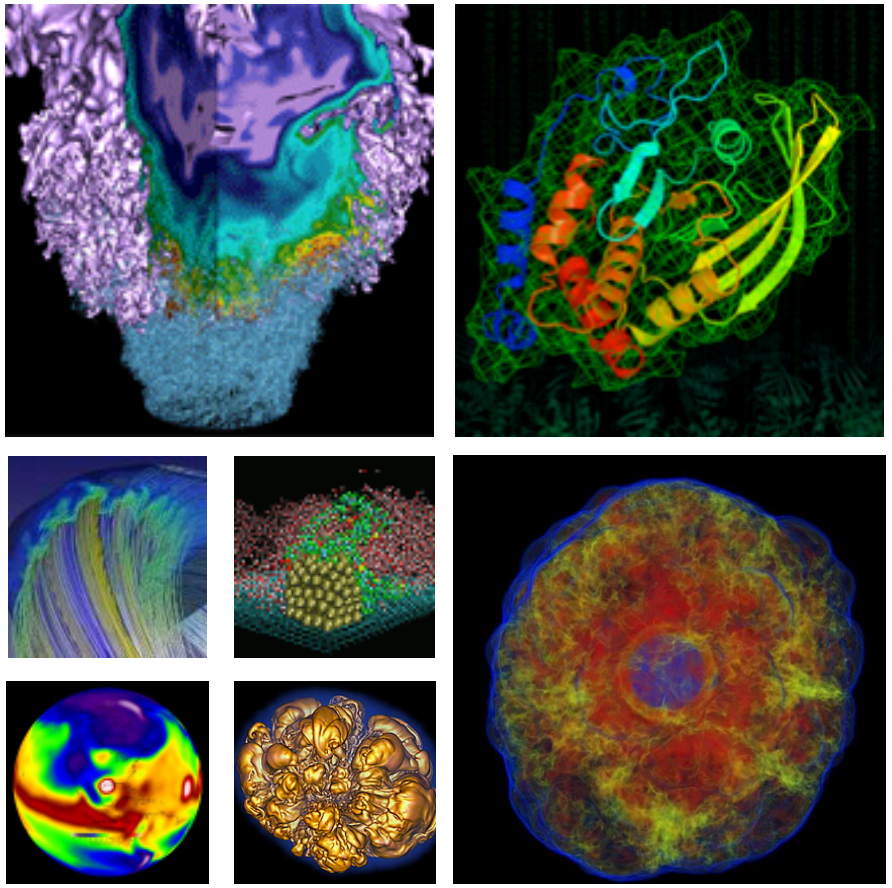
GPUs and cosmology

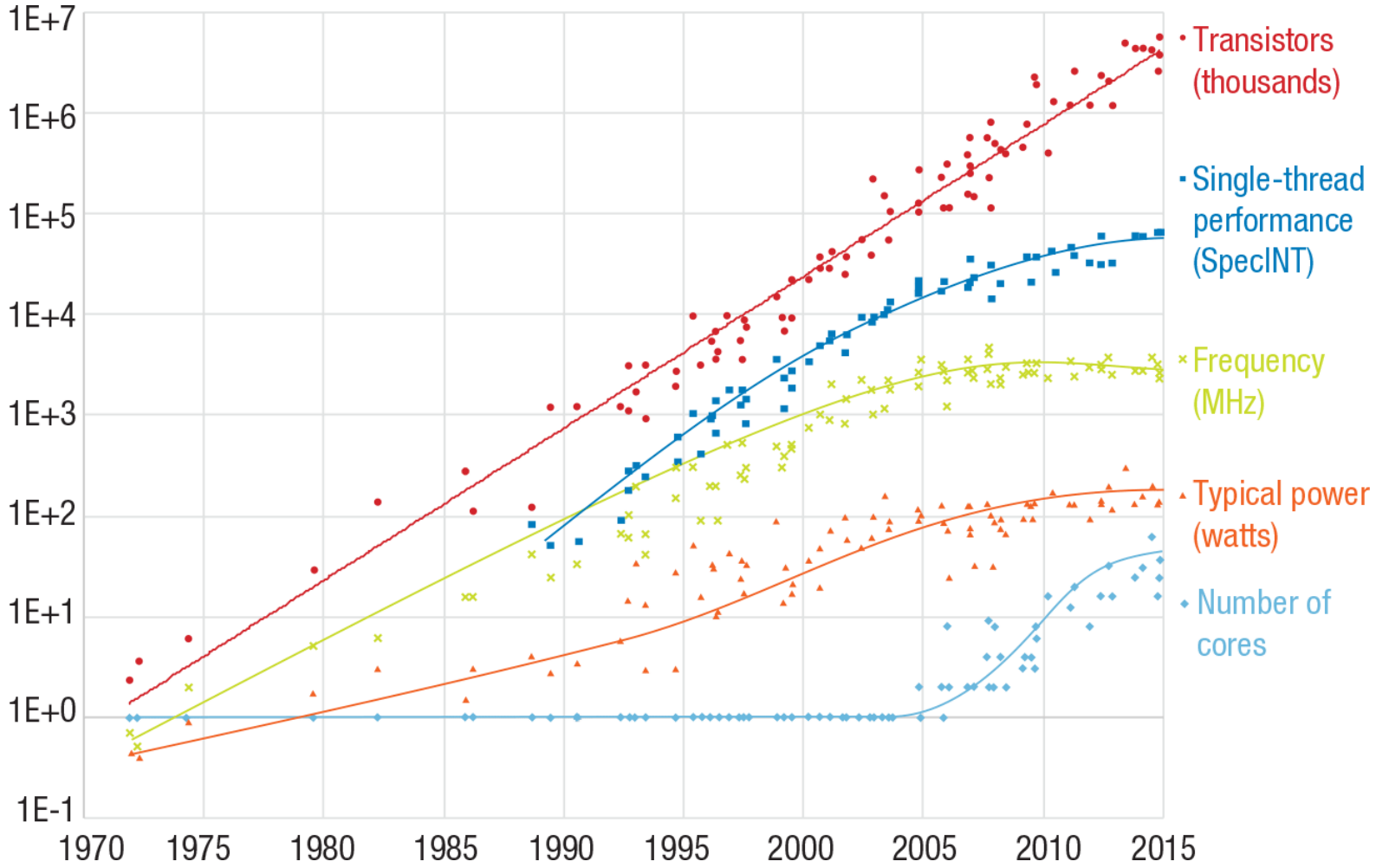


Debbie Bard
Data and Analytics Services
NERSC

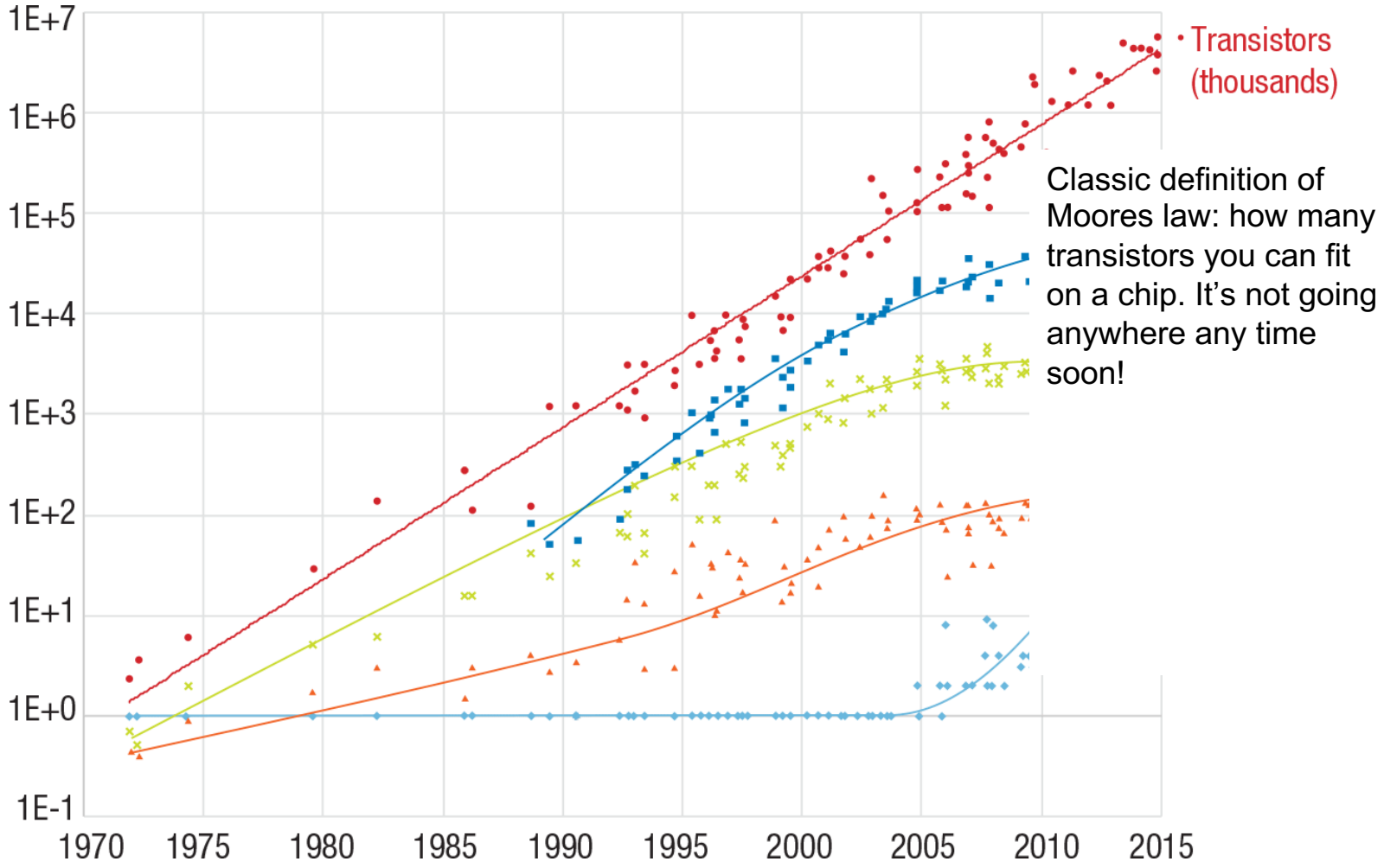
Introduction

- I'm a Big Data Architect at NERSC, in the Data and Analytics Services group
  - Formerly Project Scientist at SLAC, working on LSST
  - Formerly formerly particle physicist working on BaBar
- Interests:
  - Burst Buffer/memory hierarchy on HPC
  - Cosmology and computing (experimental)
  - HPC for Manufacturing
  - Workflow control and technology
  - Scientific data challenges

Moores law and all that

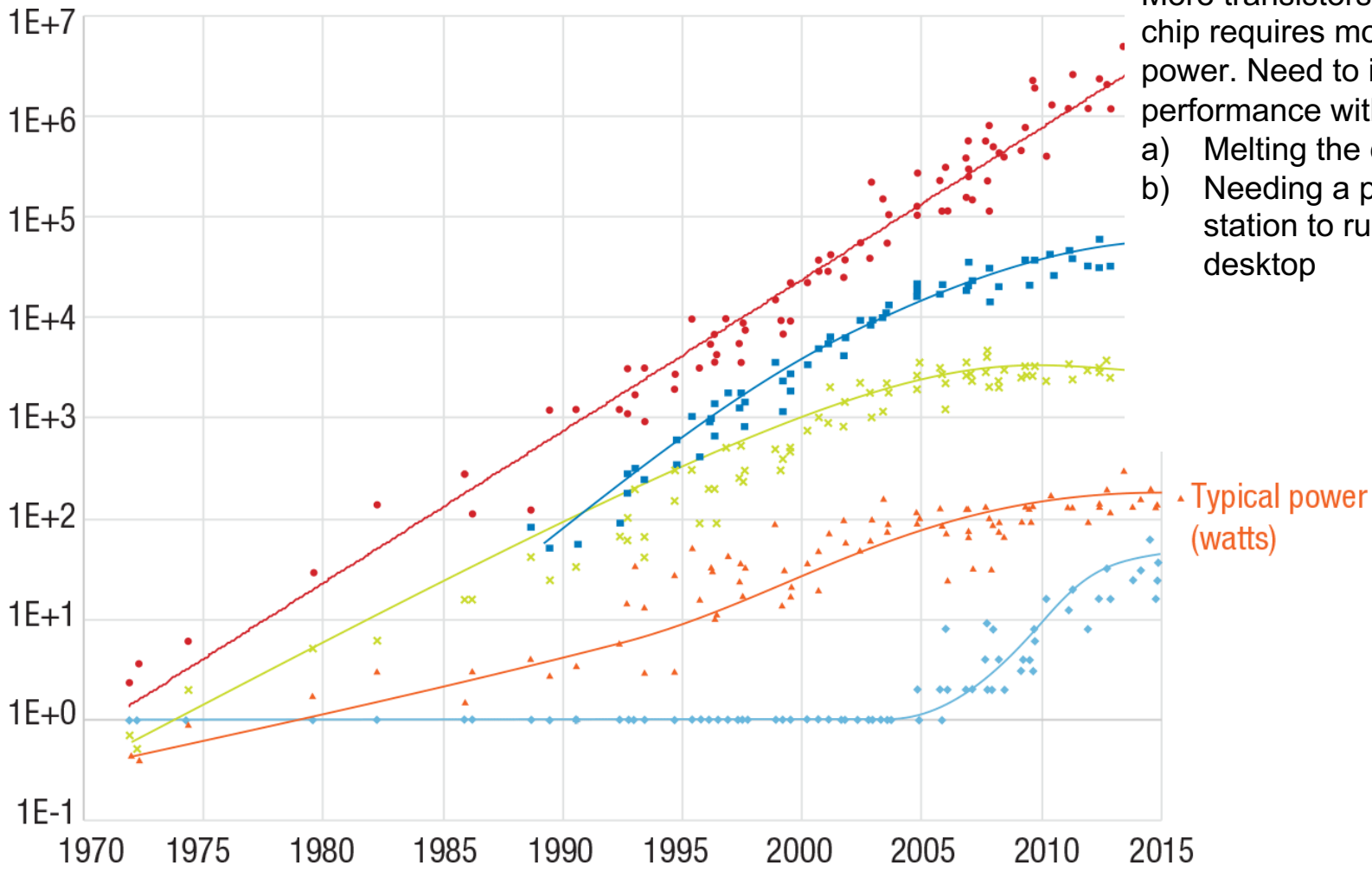


Moores law and all that



Classic definition of Moores law: how many transistors you can fit on a chip. It's not going anywhere any time soon!

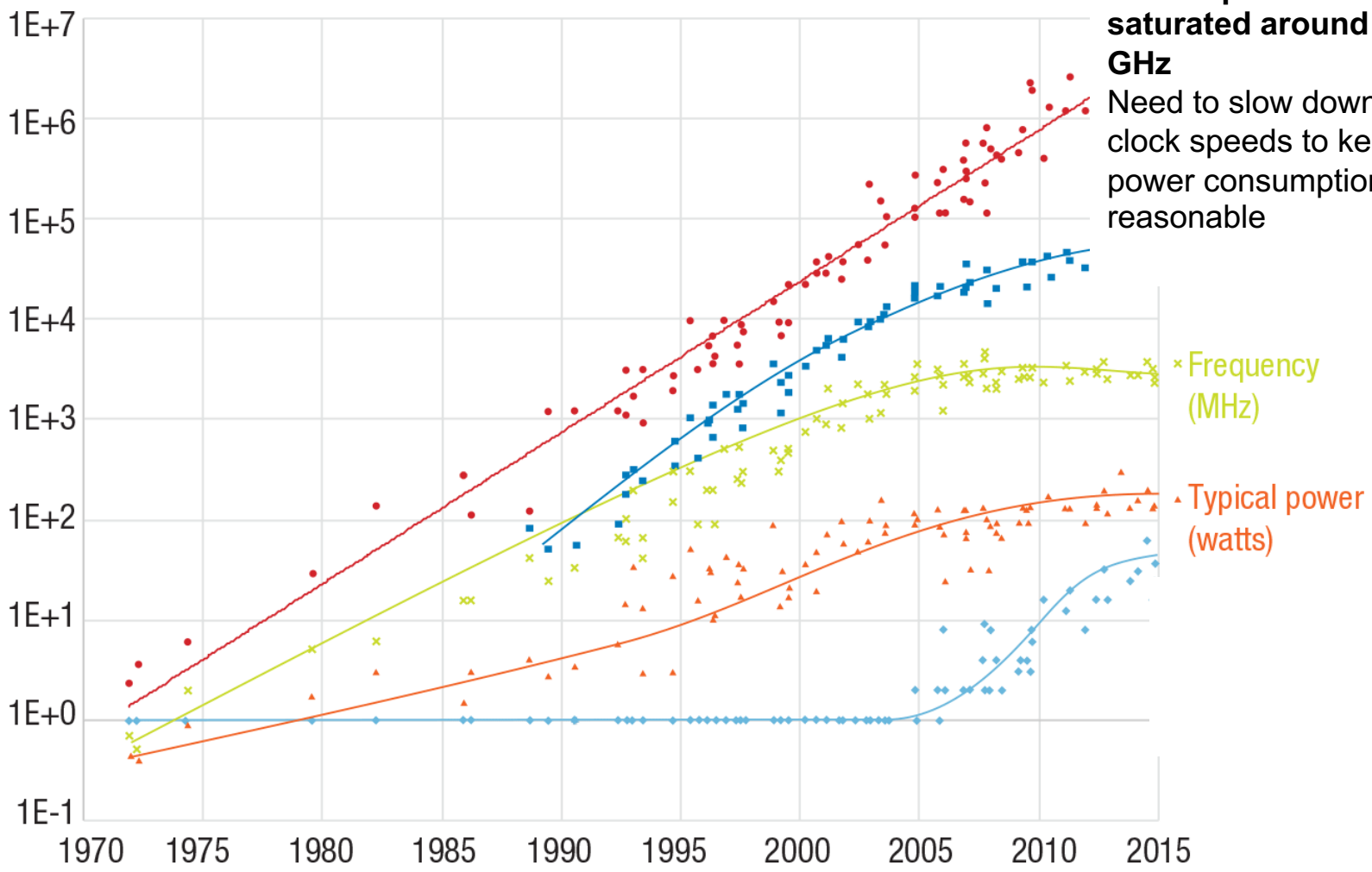
Moores law and all that



More transistors on a chip requires more power. Need to improve performance without

a) Melting the chip
b) Needing a power station to run your desktop

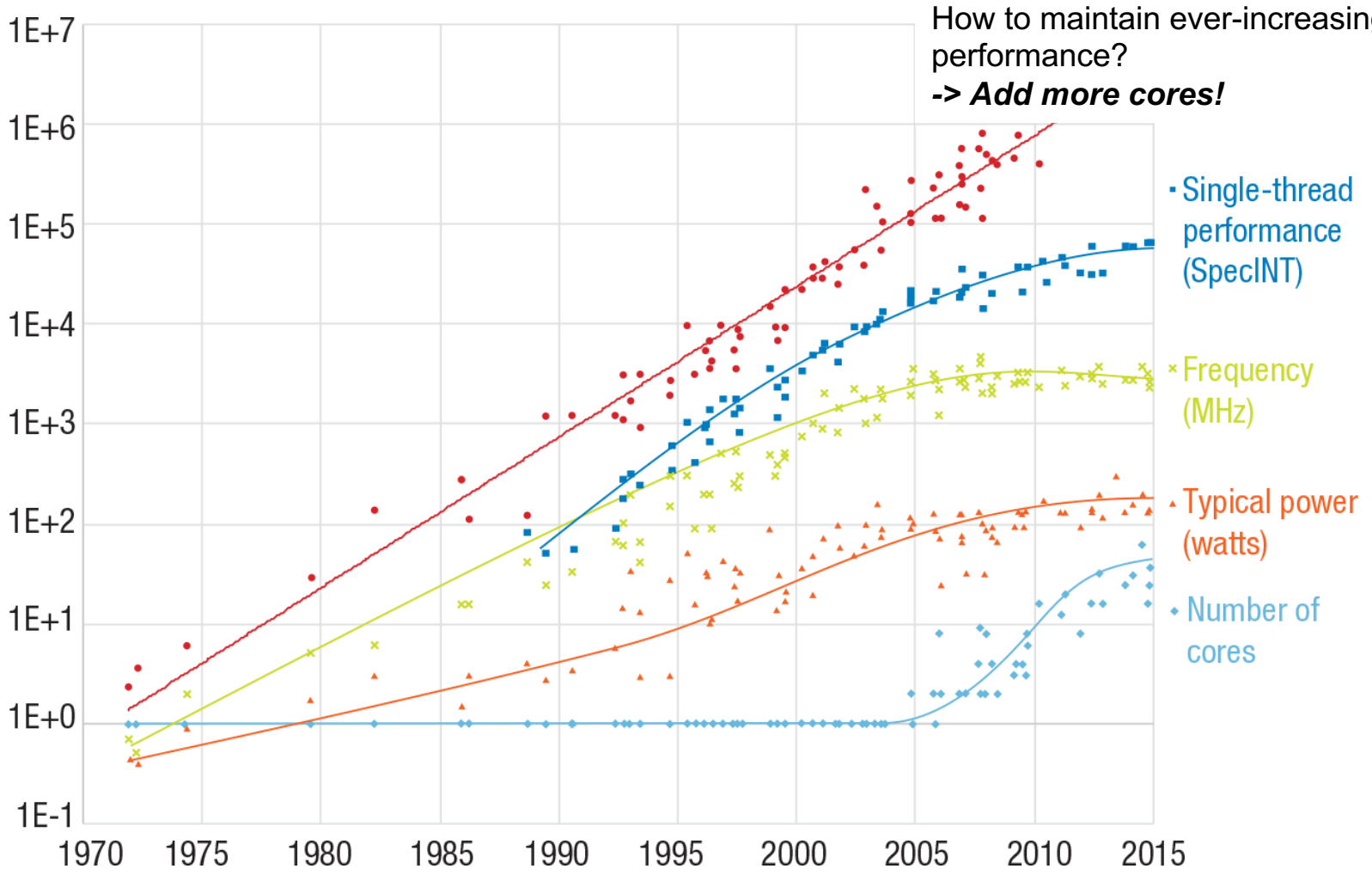
Moores law and all that



Clock speed has saturated around 3-4 GHz

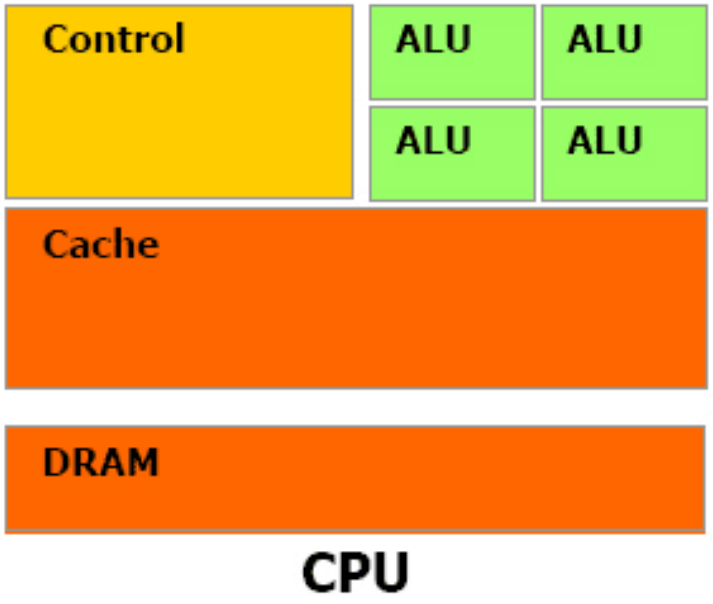
Need to slow down the clock speeds to keep the power consumption reasonable

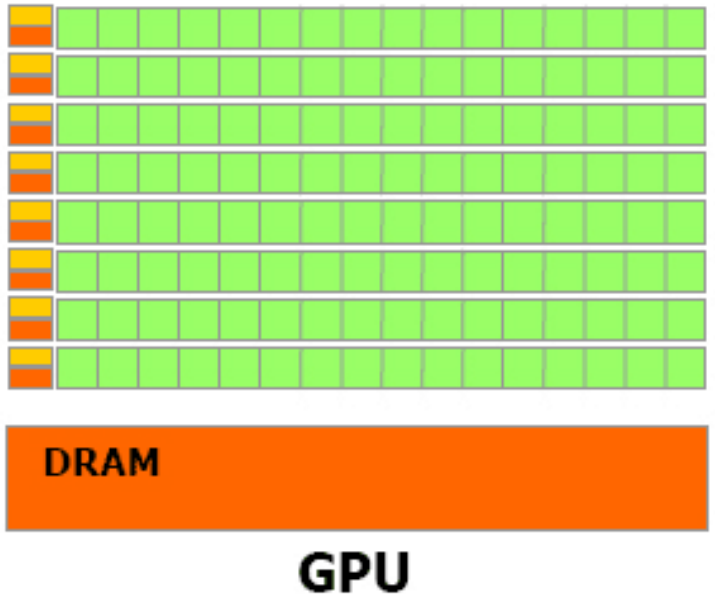
Moores law and all that



GPUs are the logical conclusion

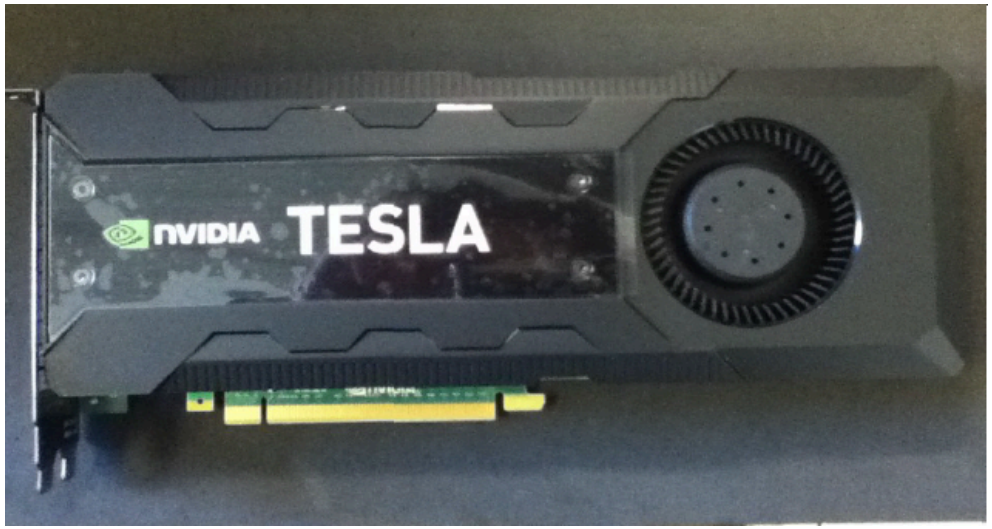
- CPU: fast cache, branching code
  - –Great for *tasks* in parallel
- GPU has loads of ALUs, fast on-chip memory
  - –Great for *calculations* in parallel

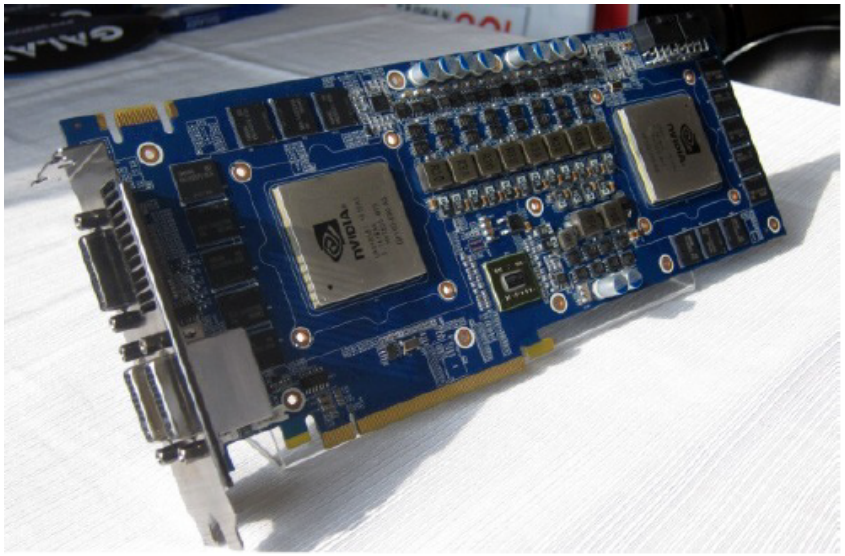




GPUs are the logical conclusion

- CPU: fast cache, branching code
  - –Great for *tasks* in parallel
- GPU has loads of ALUs, fast on-chip memory
  - –Great for *calculations* in parallel





GPUs are the logical conclusion

- CPU: fast cache, branching code
  - –Great for *tasks* in parallel
- GPU has loads of ALUs, fast on-chip memory
  - –Great for *calculations* in parallel
- The “only” limit for GPU performance is heat dissipation
  - – GPU clocks are usually slower than they need to be to save power
  - – Same for KNL…

Where's this come from?

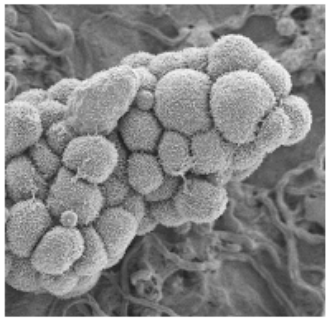
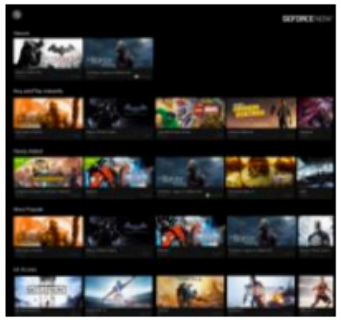
- GPU: Graphical Processing Unit
- Contains thousands of cores, can run thousands of threads -> huge economy of scale!
- Developed for fast rendering of graphics
  - –Texture, shading, movement etc.
- Designed to perform many arithmetic calculations in parallel
- Obvious applications to scientific computing
- Today the big market is Machine Learning…

GTC keynote topics

- 2012: Studying collective behaviour in insects
- 2013: Genome sequencing, car design at Chrysler
- 2014: Video games and cognitive enhancement, film production at Pixar
- 2015: DL @ Google, DL @ Baidu
- 2016: IBM Watson, car design at Toyota

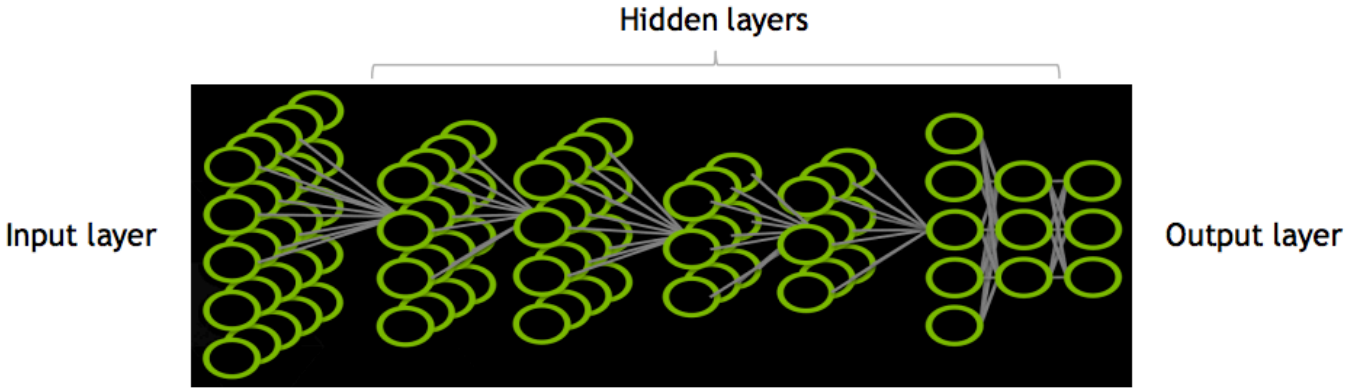
GPUs and ML

- "Deep Learning" is *everywhere*.

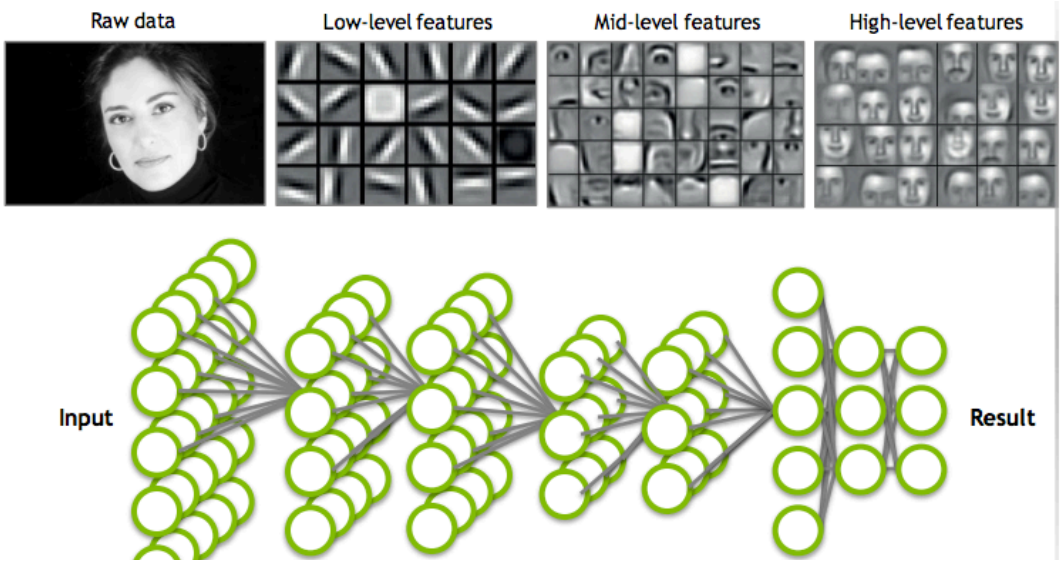




INTERNET & CLOUD	MEDICINE & BIOLOGY	MEDIA & ENTERTAINMENT	SECURITY & DEFENSE	AUTONOMOUS MACHINES
Image Classification Speech Recognition Language Translation Language Processing Sentiment Analysis Recommendation	Cancer Cell Detection Diabetic Grading Drug Discovery	Video Captioning Video Search Real Time Translation	Face Detection Video Surveillance Satellite Imagery	Pedestrian Detection Lane Tracking Recognize Traffic Sign

Why GPUs for DL?



Given sufficient training data an artificial neural network can approximate very complex functions mapping raw data to output decisions



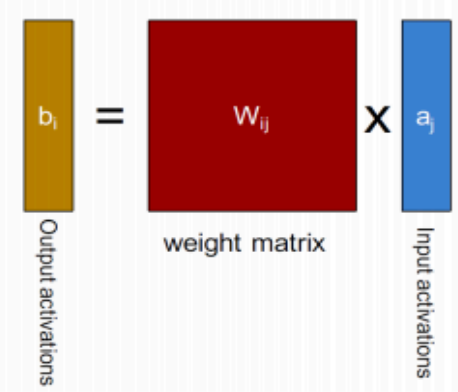
Application components:

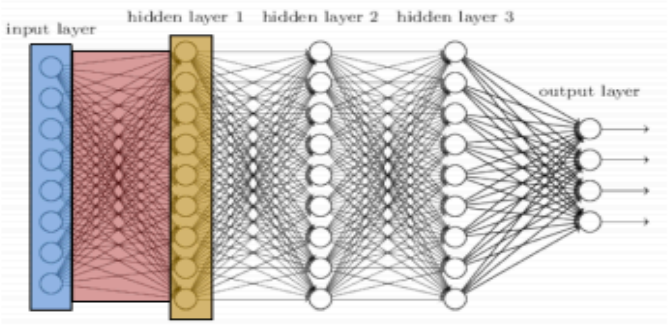
Task objective
e.g. Identify face
Training data
10-100M images
Network architecture
~10 layers
1B parameters
Learning algorithm
~30 Exaflops
~30 GPU days

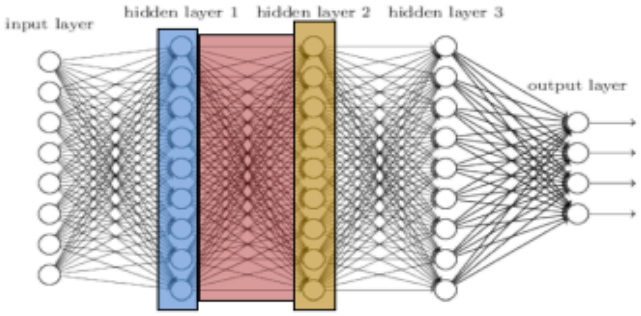
Taken from: http://on-demand.gputechconf.com/gtc/2015/webinar/deep-learning-course/intro-to-deep-learning.pdf

Why GPUs for DL?

Key operation is dense M x V







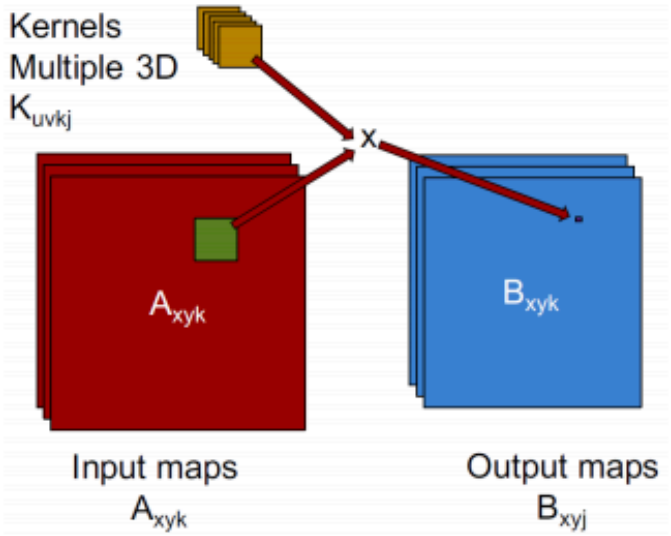
Repeat for each layer

Backpropagation uses dense matrix-matrix multiply starting from softmax scores

- There are a tremendous amount of matrix-matrix multiplications in DL algorithms – ideal for GPUs

Taken from: http://on-demand.gputechconf.com/gtc/2015/webinar/deep-learning-course/intro-to-deep-learning.pdf

Why GPUs for CNN?



Filters conserved through plane

Multiply limited - even without batching.

6D Loop

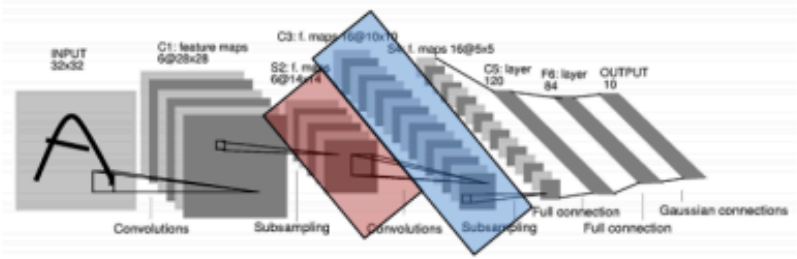
For each output map j

For each input map k

For each pixel x,y

For each kernel element u,v

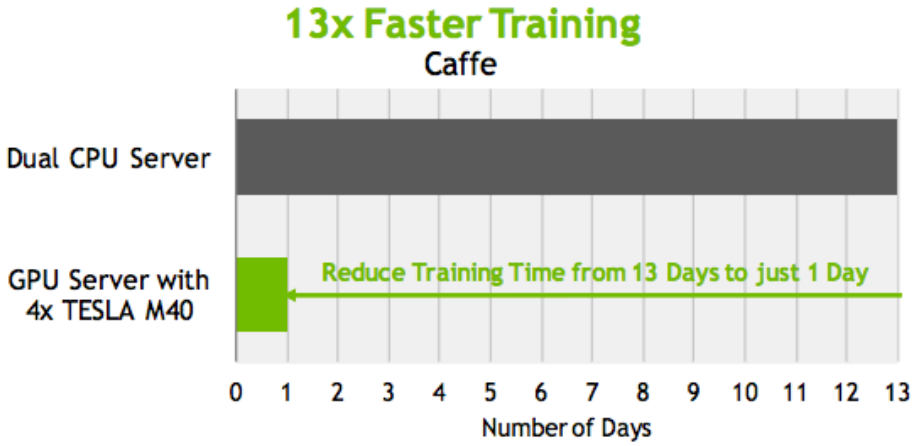
B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}



- There are a tremendous amount of matrix-matrix multiplications in CNN algorithms – ideal for GPUs

Taken from: http://on-demand.gputechconf.com/gtc/2015/webinar/deep-learning-course/intro-to-deep-learning.pdf

GPUs are *very* fast for DL



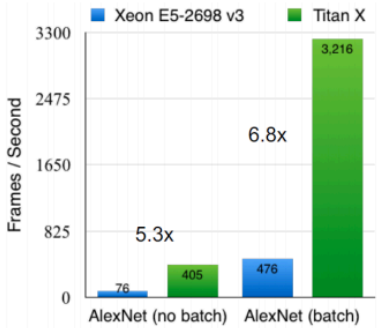
- Can reduce training time by an order of magnitude – useful if it takes 2 weeks to train your network.

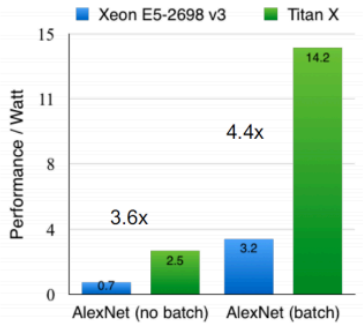
CUDA Cores	3072
Peak SP	7 TFLOPS
GDDR5 Memory	12 GB
Bandwidth	288 GB/s
Power	250W

Note: Caffe benchmark with AlexNet,
CPU server uses 2x E5-2680v3 12 Core 2.5GHz CPU, 128GB System Memory, Ubuntu 14.0

Comparing CPU and GPU - server class

Xeon E5-2698 and Tesla M40

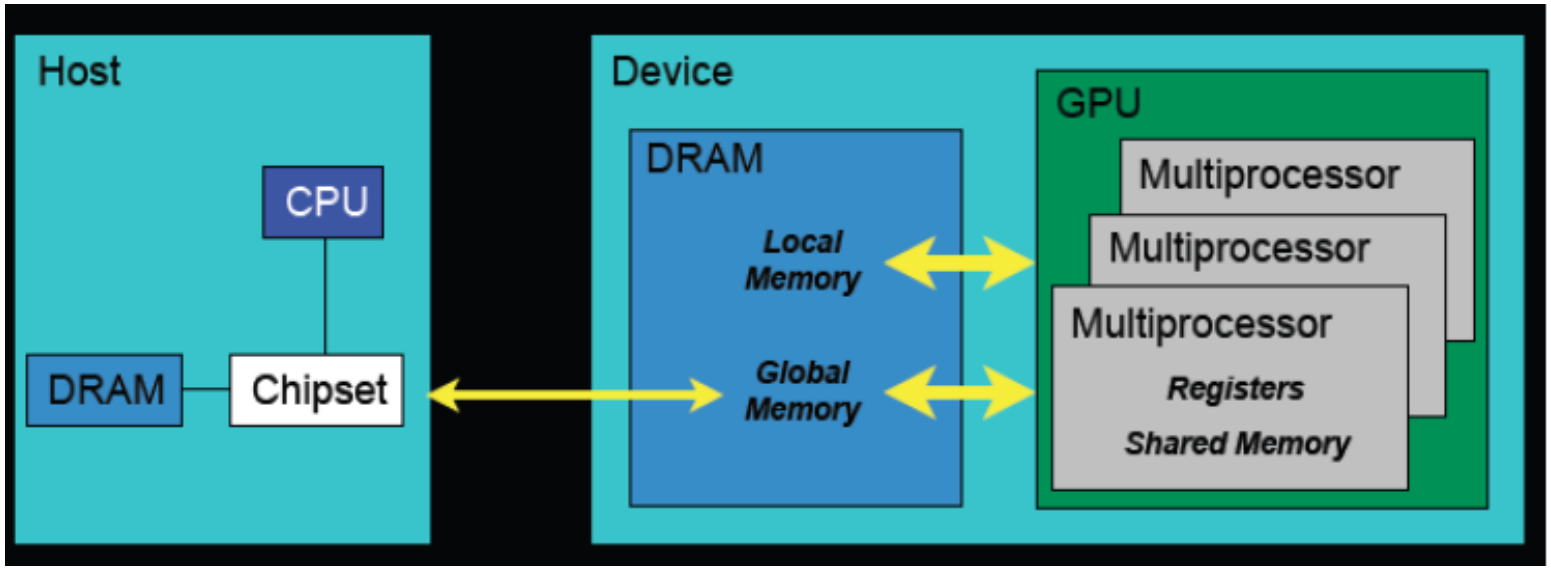




Taken from: http://on-demand.gputechconf.com/gtc/2015/webinar/deep-learning-course/intro-to-deep-learning.pdf

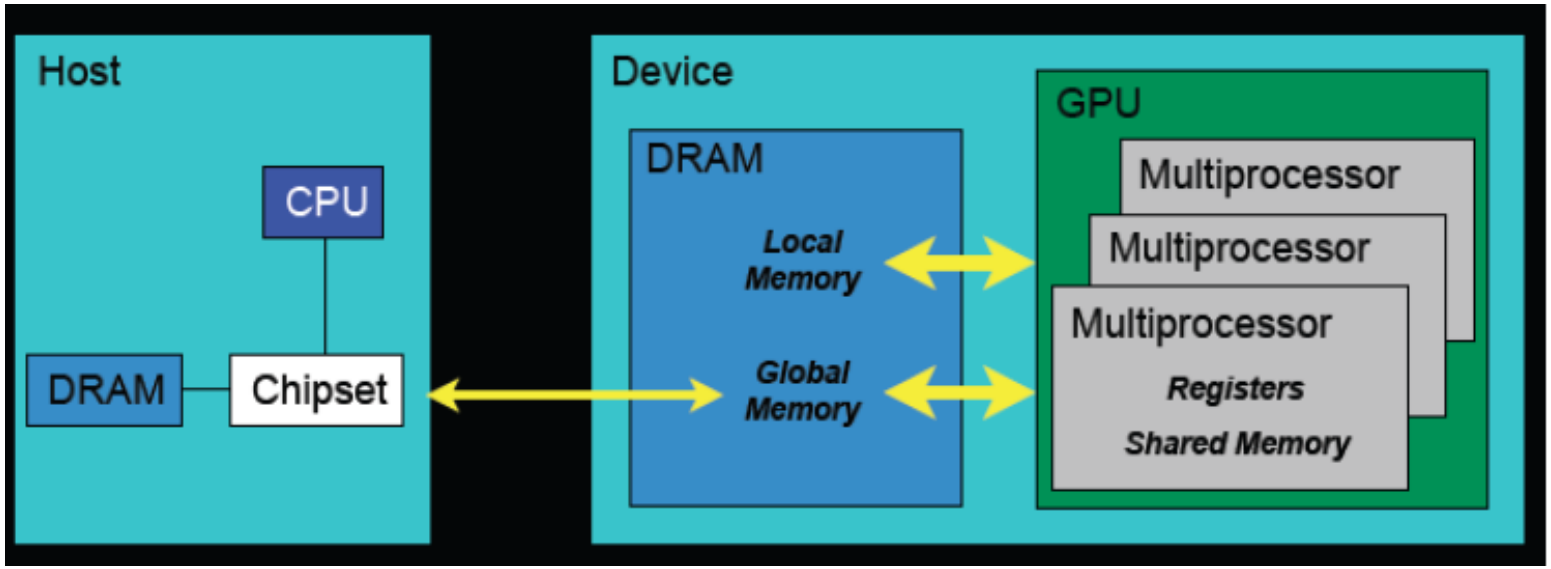
DL aside, why use GPUs for *science*?

- GPGPU: General Purpose GPU computing.
- In an ideal system, run sequential part of the app on the CPU while the GPU handles the computationally-intensive part



DL aside, why use GPUs for *science*?

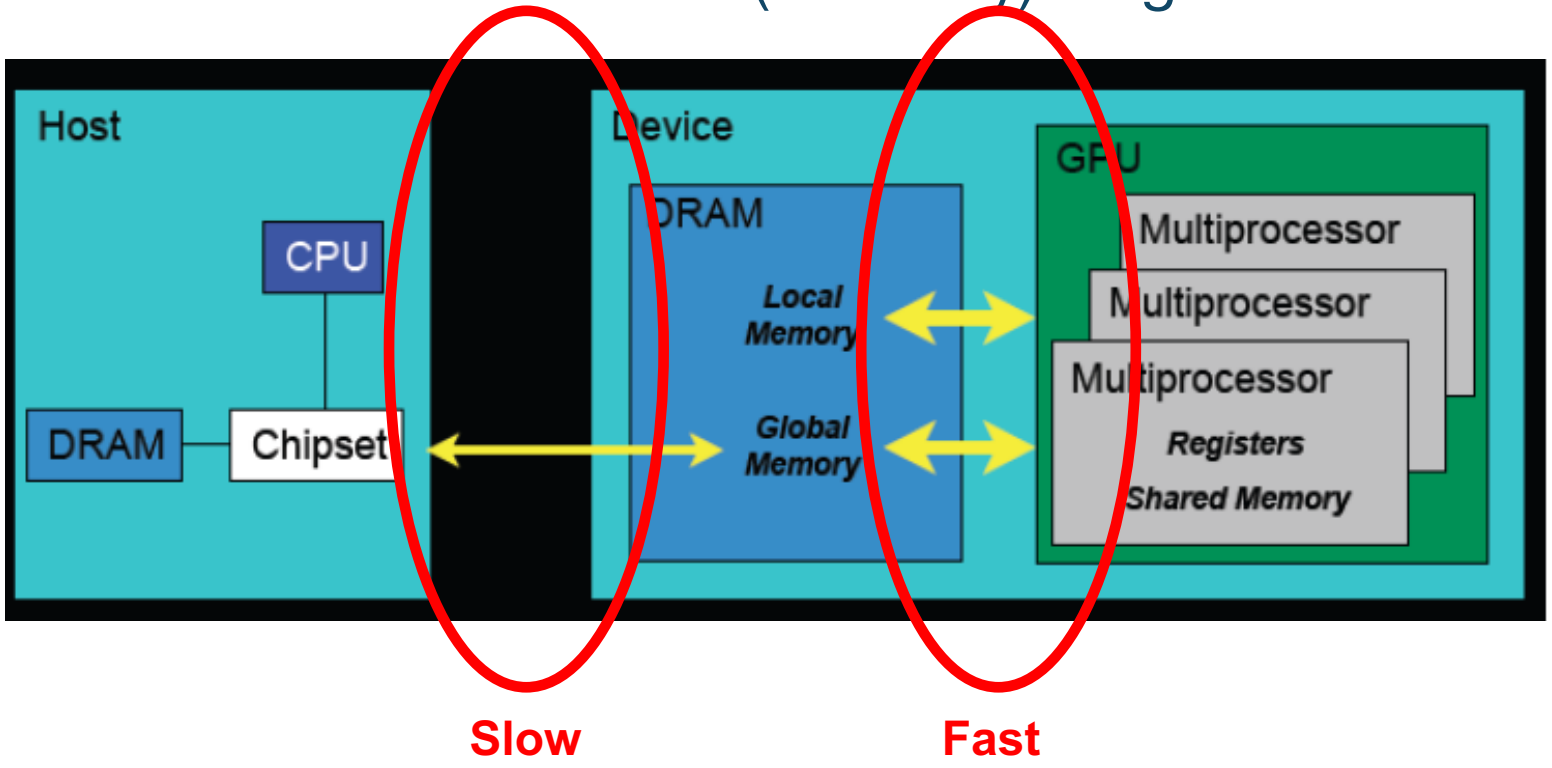
- GPGPU: General Purpose GPU computing.
- In an ideal system, run sequential part of the app on the CPU while the GPU handles the computationally-intensive part



- Cosmology N-Body sims, protein folding, fluid dynamics, QCD calculations, weather modelling, bioinformatics….

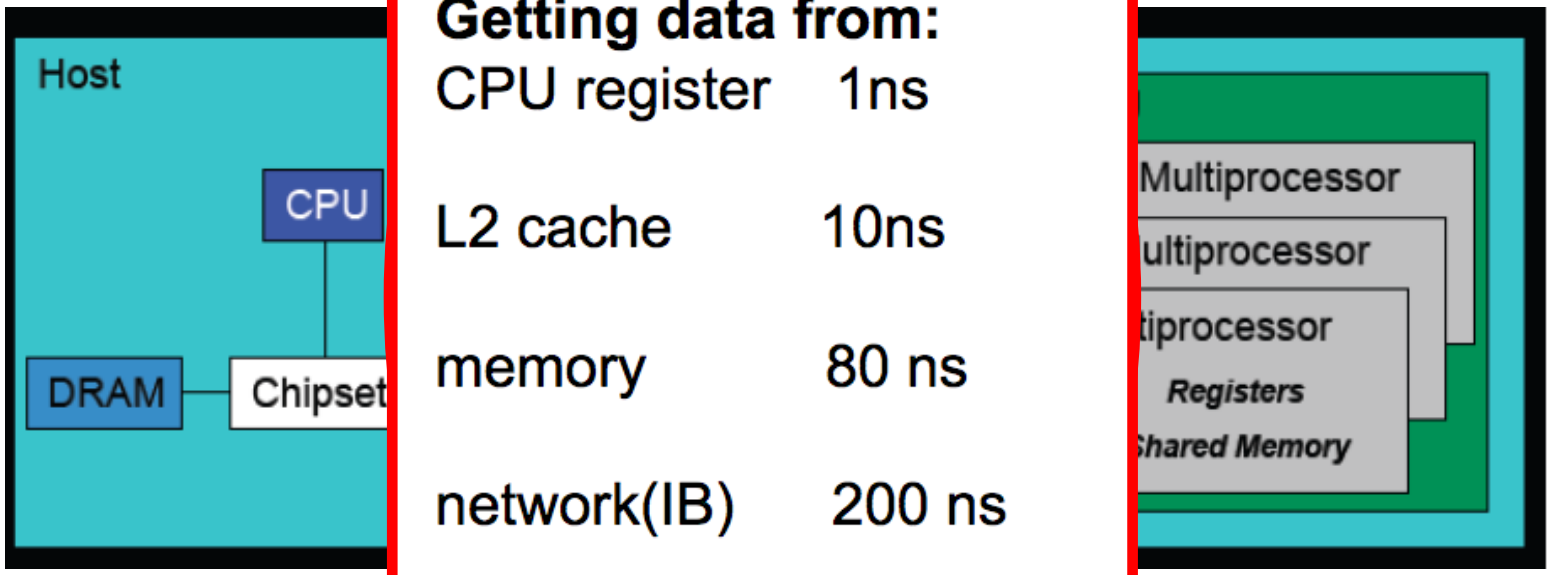
GPUs won't solve everything

- Not a panacea!
- I/O is by far the slowest part – transferring data to/from the GPU device can take a (relatively) long time



GPUs won't solve everything

- Not a panacea!
- I/O is by far the slowest part – transferring data to/from the GPU device can take a (relatively) long time



Getting data from:

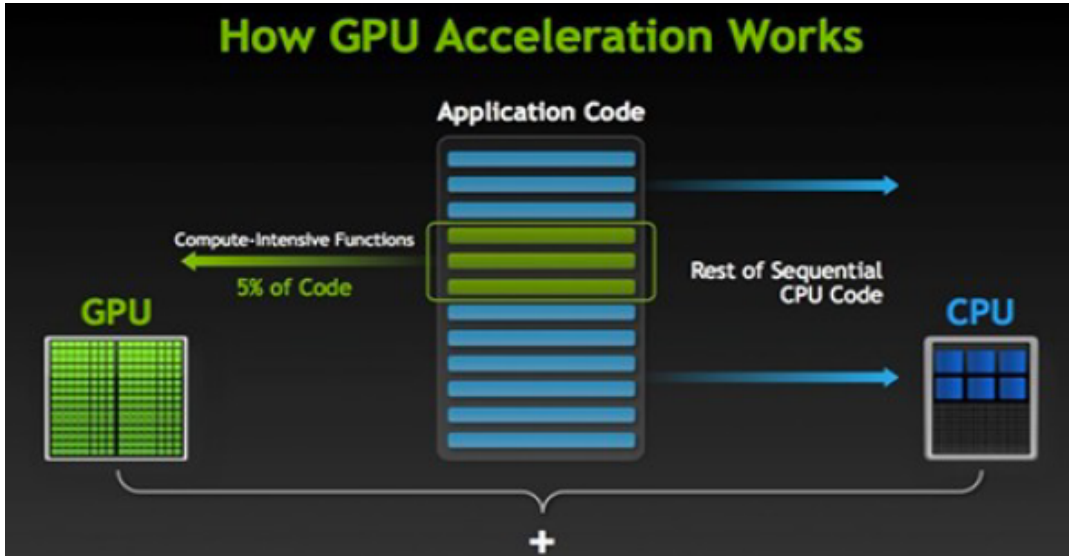
CPU register	1ns
L2 cache	10ns
memory	80 ns
network(IB)	200 ns
GPU(PCIe)	50.000 ns
harddisk	500.000 ns

GPUs won't solve everything

- Not a panacea!
- I/O is by far the slowest part – transferring data to/from the GPU device can take a (relatively) long time
- Memory on device is limited
- Math calculations are performed differently on the GPU – optimised for fastest performance so precision may be an issue
  - Nvidia has been very responsive in improving this – full precision is available but it's not as performant.
  - Only the highest-end GPUs can handle double precision/ECC.

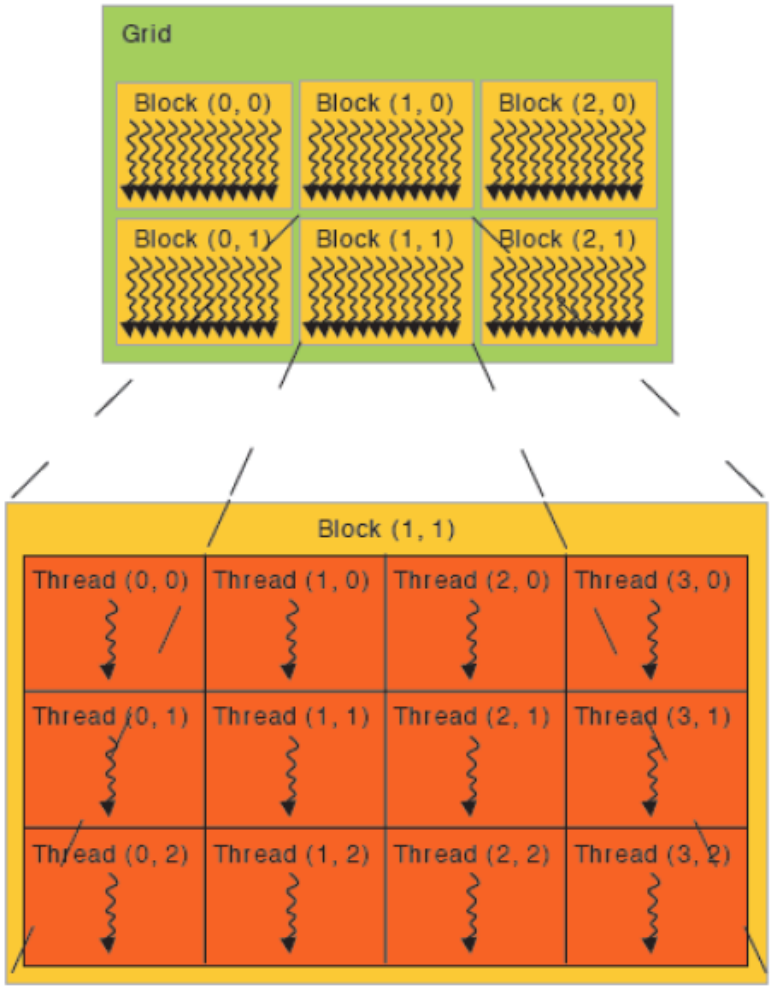
GPGPU

- CPU and GPU can be used together for maximal efficiency.
- Sequential part on CPU, computationally-intensive part on GPU
  - Offloading
  - Compare to “traditional” HPC: c/c++/Fortran + libraries
- Aim for **maximum occupancy** of both CPU and GPU: ideal if you can use both simultaneously!



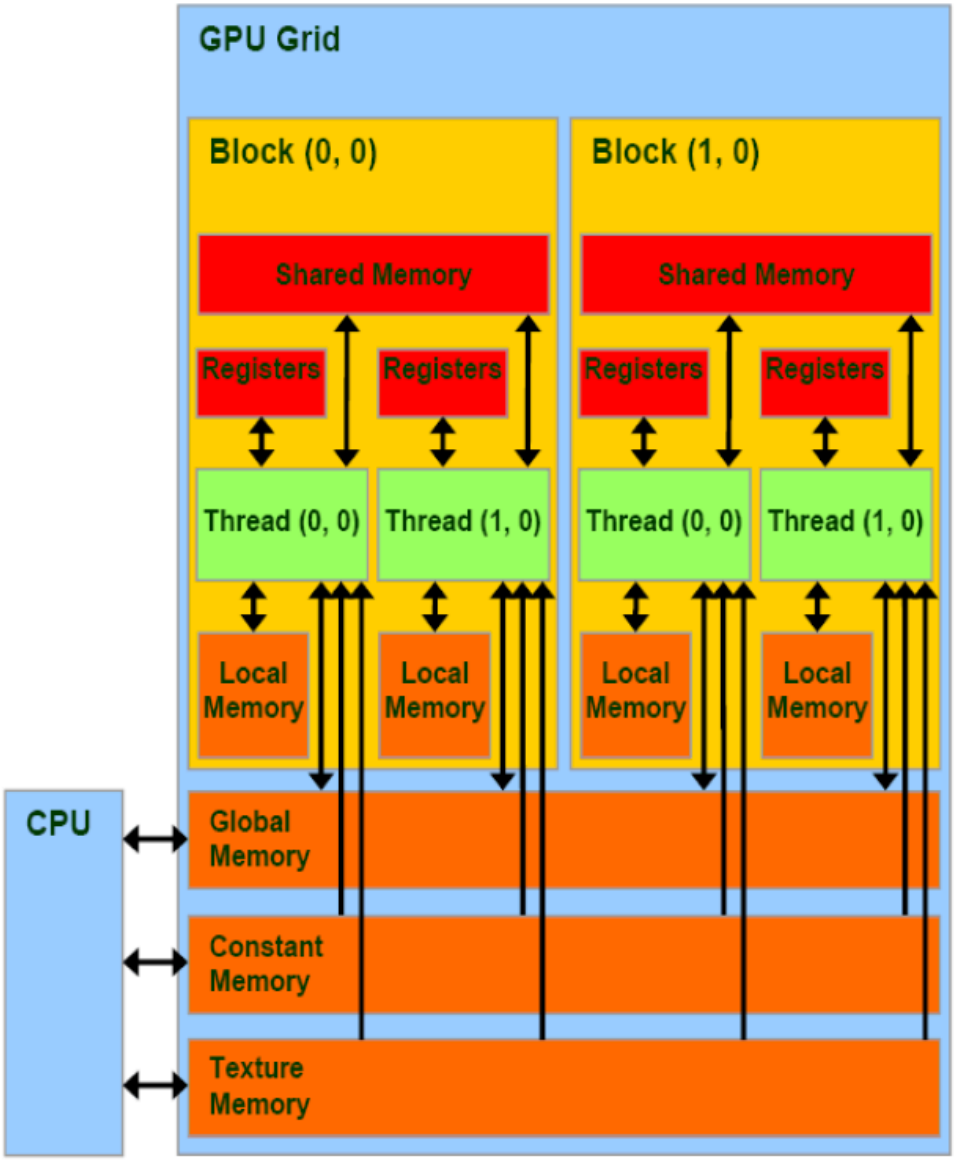
GPU Terminology

- Launch a compute **kernel**
- Kernel executes *same code* for each **thread**
- Threads are launched in **blocks**
- Blocks are arranged in a **grid**
- Threads in a block can share memory and synch execution, but each thread is independent
- Each thread has a unique **index**



GPU memory

- GPU has on-board global and shared memory
- **Global**: slowest, largest (12-16GB)
- **Shared/local**: faster, smaller, user-programmable block cache (48KB)
- **Constant/texture** cache: register-fast, constant (64 KB)



GPU memory

Getting data from:

CPU register 1ns

L2 cache 10ns

memory 80 ns

network(IB) 200 ns

GPU(PCIe) 50.000 ns

harddisk 500.000 ns

Memory	Location on/off chip	Cached	Access	Scope	Lifetime
Register	On	n/a	R/W	1 thread	Thread
Local	Off	†	R/W	1 thread	Thread
Shared	On	n/a	R/W	All threads in block	Block
Global	Off	†	R/W	All threads + host	Host allocation
Constant	Off	Yes	R	All threads + host	Host allocation
Texture	Off	Yes	R	All threads + host	Host allocation

Programming GPUs

- I've only ever done this directly, using CUDA
- But these days, you can script for the GPU perfectly easily in python/R/etc.

	SMP multicore parallelism	MMP massive parallel processing	GPGPU CUDA openCL
R	doMC, doSMP, pnmath, BLAS **no max cores**	doSNOW, doMPI, doRedis	rgpu, gputools
python	multiprocessing futures	parallel python, mpi4py	pyCUDA, pyOpenCL
MATLAB	parfor, spmd **max 8 cores**	jobs, pmode	gpuArray

Simple example

- Example kernel to add matrices

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Why might GPUs be useful/exciting?

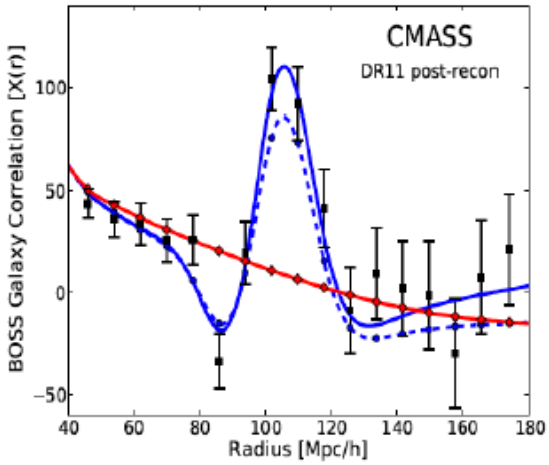
- Run more problems in the same time
  - Explore parameter space
- Solve a bigger problem in the same time
  - Improve resolution
- Solve a more complex problem in the same time
  - Get a more accurate solution
- Potential TFLOPS on your desktop
  - "desktop supercomputing"
  - don't need to wait/fight for time on Cori…
- Quick turnaround allows me to play with ideas
- Real-time detection of events

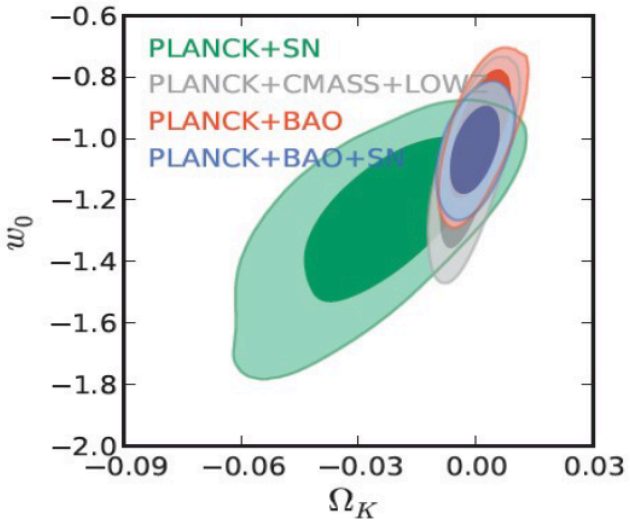
Why GPUS in cosmology?

- Many problems in cosmology are data-parallel
- We have large amounts of data, and even more simulations
- Lots of our calculations don't scale to large datasets
  - –E.g. correlation functions
- Approximation functions are useful (e.g. k-d trees) but can introduce systematic errors
  - – What will we do in the era of "precision cosmology"?
- Cosmologists needs to be data scientists…..

Two-point correlation function

- Galaxies are good tracers of the matter concentration in the universe
- 2PCF: measure of over-density of galaxy pairs at different separations, compared to a random distribution
- Can be used to constrain models of cosmology





2PCF

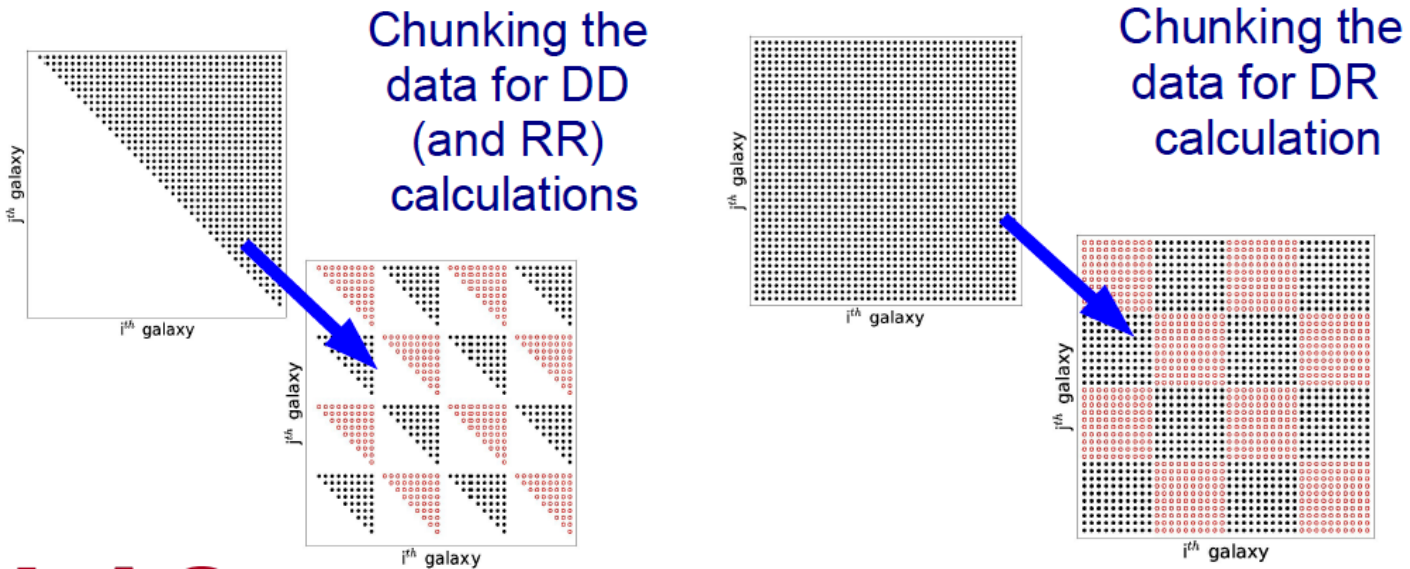
- Galaxy 2PCF can be calculated using pair counts with the Landay-Szalay estimator:

$$w(\theta) = \frac{DD - 2DR + RR}{RR}$$

- Where DD is data-data pair counts, DR is data-random pair counts etc, and theta is the angular distance.
- O(N2) calculation
  - Ideal for parallelism on the GPU!
  - Ponce++ (2012), Bard & Bellis (2013)

2PCF on the GPU

- Calculating the distance between two galaxies is a pretty cheap calculation – how to keep the threads busy?
  - –Don't want to do one-pair-per-thread because of kernel launch overhead
- Split data and calculations into chunks
  - –Each chunk is a kernel call on on (or more) GPUs
    - Embarrassingly/pleasingly parallel



2PCF on the GPU

- Compute time for DD pair count reduced by a factor ~140 compared to single thread on CPU
  - –of course, that's a false comparison because you'd never use a single thread on the CPU….. Would you?

1. Copy vectors of galaxy positions to the GPU global memory.
2. Determine size and position of submatrices of the calculation.
3. Launch kernel for each submatrix:
   Allocate local memory for histogramming on each block. Each thread calculates the angular separation for a column of the submatrix.
4. Sum the local memory histogram arrays in global memory.
5 Copy the histogram arrays back to the CPU and sum them.
6. Continue to loop over all the submatrices until the entire calculation is done.

	10^4 galaxies	10^5 galaxies	10^6 galaxies
CPU	24s	1305s	231,225s
GPU	0.2s	16s	1254s

2PCF on the GPU

- What about memory issues?
  - Can fit entire dataset on GPU global memory up to 12GB (~5M galaxies)
  - Fit histogram (of galaxy separations) in shared memory for faster access
- Bus speed can be an issue when transferring large amounts of data to/from the GPU
  - Ideally, **overlap data transfer and compute**
  - Transfer data for next sub-matrix to the GPU global memory while the previous sub-matrix is using GPU registers for calculation
  - Very efficient use of GPU IO and compute!

Want to look at some simple CUDA?

- https://github.com/djbard/ccogs
- Points to note:
  - explicitly malloc on the GPU

```
cudaMalloc((void **) &d_alpha0, size_of_galaxy_array0 );
cudaMemset(d_alpha0,0,size_of_galaxy_array0);
```

  - explicitly transfer data over

```
cudaMemcpy(hist, dev_hist, size_hist_bytes, cudaMemcpyDeviceToHost);
```

  - think about how to split calculation into grids/blocks

```
distance<<<grid,block>>>(d_alpha0, d_delta0,d_alpha1, d_delta1, x, y, max_x, max_y, dev_hist, hist_lo
```

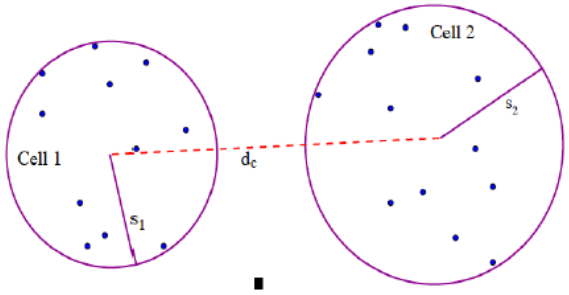
  - explicitly transfer result back

```
cudaMemcpy(d_alpha0, h_alpha0, size_of_galaxy_array0, cudaMemcpyHostToDevice );
```

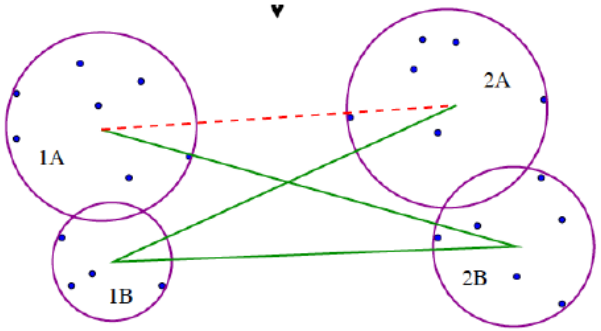
  - **not very complicated, but you *do* need to think about memory carefully.**

What about using a k-d tree?

- k-d tree spits data into progressively smaller cells, take centroid of galaxy location in smallest cell
  - O(N1*N2) -> O(N1+N2).
  - Size of smallest cell is tunable



- Two cells. Red line indicates cell size (s) is too large compared to distance between them (d) so they must be subdivided. $\frac{s_1 + s_2}{d_c} < b$



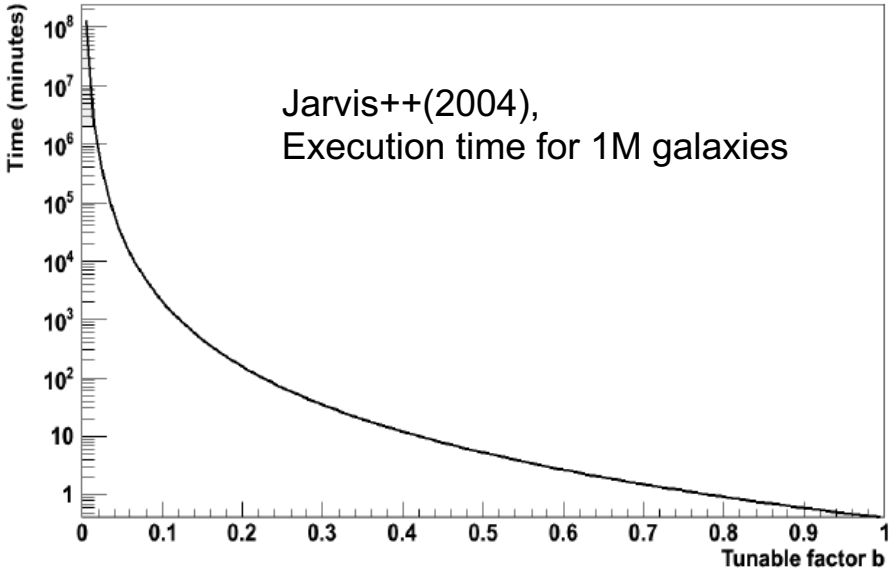
- Cells are sub-divided. Green line indicates where cell size (s) is small enough compared to distance (d).

- Binning the dataset in angular separation effectively smooths the 2PCF as we ignore information within each bin.

Image from Jarvis++(2004)

k-d tree vs GPU

- Key is tunable factor. If small, calculation becomes exact but very slow. If large, uncertainties become untenable.
- Reasonable value depends on your requirements.

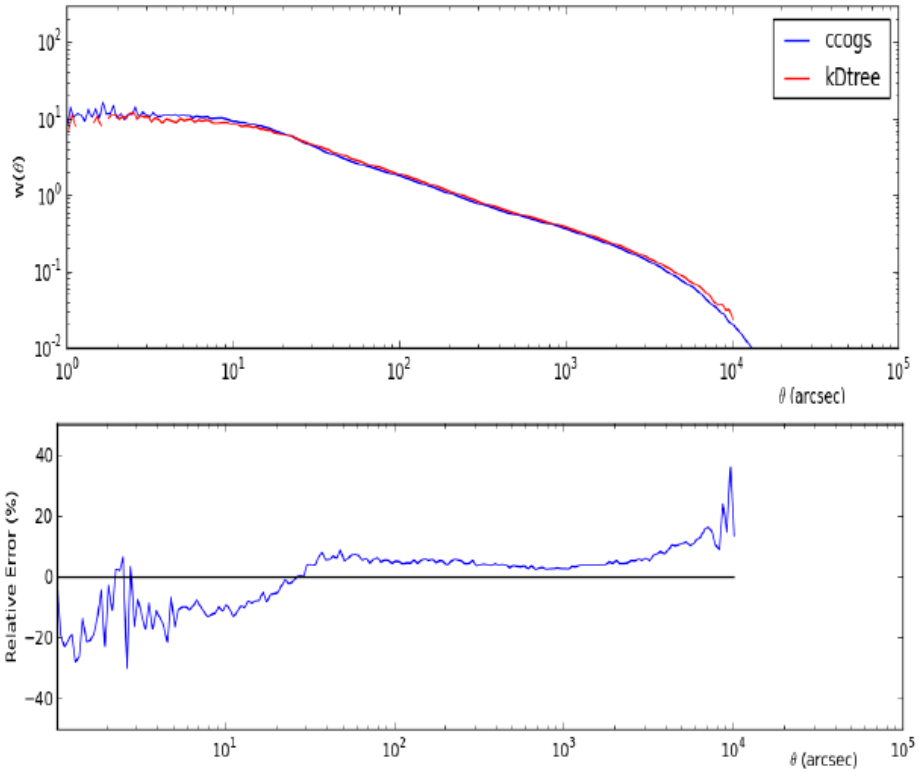


- What level of uncertainty can you tolerate for “precision cosmology”?

Large Synoptic Survey Telescope

k-d tree vs GPU

- Key is tunable factor. If small, calculation becomes exact but very slow. If large, uncertainties become untenable.
- Reasonable value depends on your requirements.

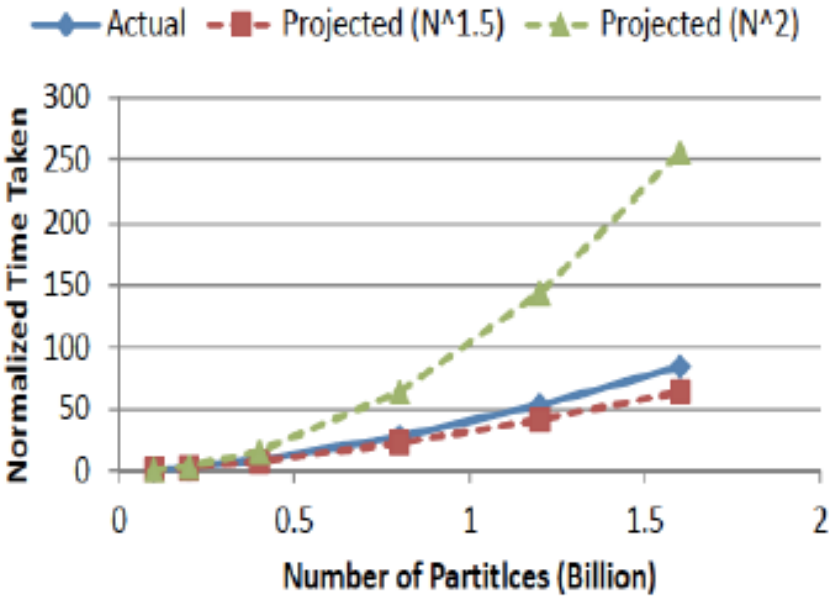


- E.g. k-d tree is 5x faster than full calculation, but has ~10% errors!
- K-d tree: 94min
- GPU: 460 min
- 1.5M data, 5M random galaxy dataset

What about HPC?

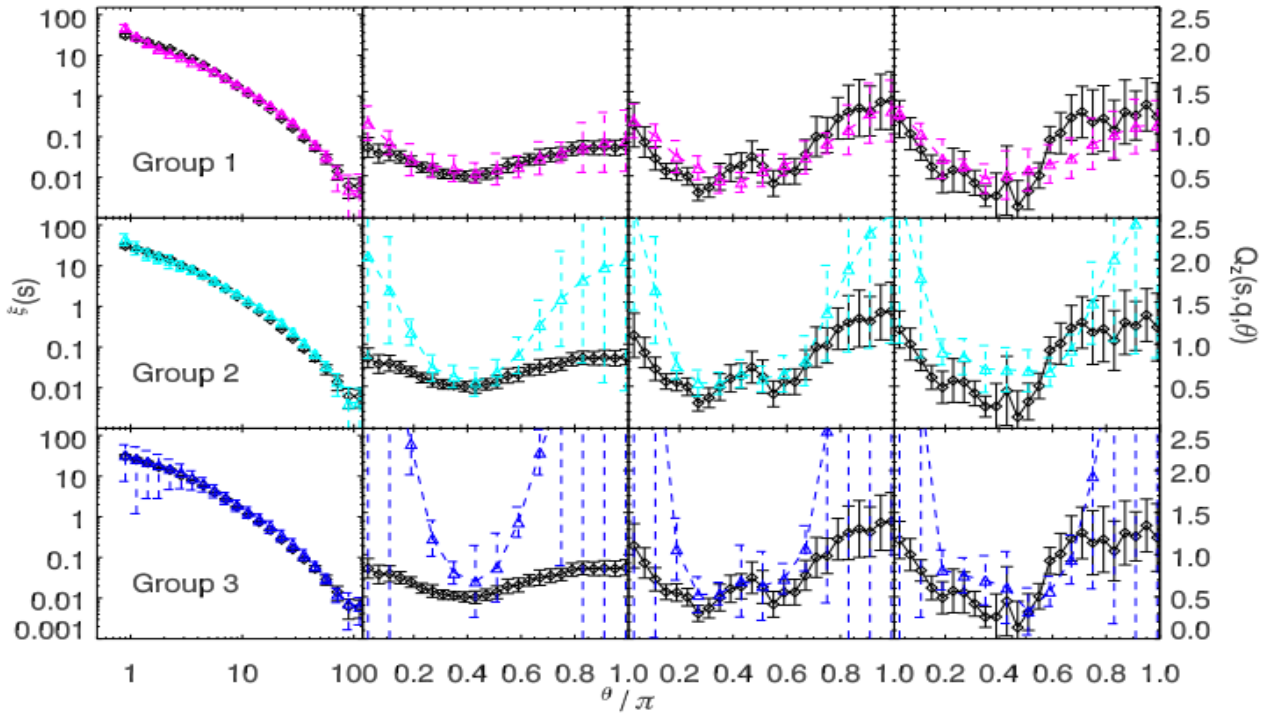
- Chhugami++(2012) made a maximally efficient, load-balanced k-d tree on a supercomputer
  - –Histogramming doesn't work well with SIMD architecture…

- 2PCF for 1.7 billion particles in 5.3 hours
  - – 25.6k cores 1.06 Pflops.
  - – Achieved $O(N^{1.5})$ scaling.
  - – Unclear how much uncertainty is introduced



2PCF is not the end of the story…

- 3PCF can provide information about non-Gaussianity in the early universe inaccessible by the 2PCF
- Theoretical models that match the 2PCF can be ruled out by the 3PCF.



From Kulkani++(2009)

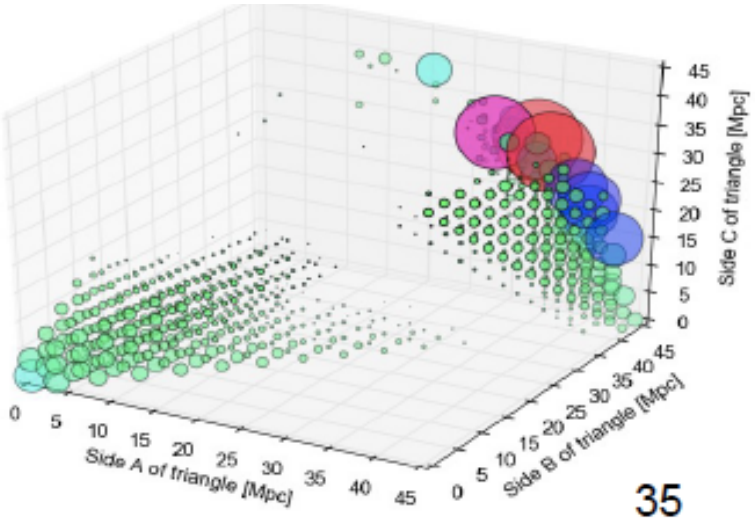
3PCF

- Use Szapudi & Szalay estimator:

$$\zeta = \frac{DDD - 3DDR + 3DRR - RRR}{RRR}$$

- Count triplets and bin according to triangle parameters
  - 3D histogram of results

- Sample 3D histogram output. Size and colour of sphere represents occupancy of the voxel.
- Note high load imbalance!



3PCF on the GPU

- Our approach: split dataset into sub-matrices
  - Each sub-matrix of data is sent otthe GPU
  - One thread in the kernel sums the @ triplets for one galaxy, and calculates triangle parameters
  - Histogram incremented in **shared memory** using atomicAdd.
  - Histograms summed back on CPU.
- Remember: shared memory is ~2x faster than global memory, but is small (typically ~48KB) and *only accessible from the kernel*.

Shared memory example

- In the kernel, first initialise the shared memory histogram

```
__shared__ int shared_hist[(DEFAULT_NBINS+2+2)*(DEFAULT_NBINS+2+1)*(DEFAULT_NBINS+2)/6];
// Note that we only clear things out for the first thread on each block.
if(threadIdx.x==0)
{
    for (int i=0;i<tot_hist_size;i++)
        shared_hist[i] = 0;
}
__syncthreads();
```

- Use atomicAdd to increment the histogram bin

```
atomicAdd(&shared_hist[totbin],1);
```

- Accumulate the elements of the shared histogram into the summation histogram, that will be copied back to the CPU

```
__syncthreads();

if(threadIdx.x==0)
{
    for(int i=0;i<tot_hist_size;i++)
    {
        dev_hist[i+(blockIdx.x*tot_hist_size)]=shared_hist[i];
    }
}
}
```

3PCF on the GPU: optimisations

- Use submatix size that requires maximum # threads
  - Fully occupy all ALUs on GPU
- Bin histogram as finely as possible, to avoid serialisation in atomicAdd
  - Don't want many threads trying to write to the same memory address!
- If desired histogram bins don't fit into shared memory, have to use global memory
  - Global memory much slower.
- Stream data transfer and kernel execution
  - Can transfer data to GPU, from GPU, and run calculation all concurrently.

https://github.com/mattbellis/3pcf/blob/master/src/3pcf_Bellis.cu

Cuda Streaming: code example

```
//Creating 4 streams, each assigns a local thread index to the array
//
#include <stdio.h>
#include <stdlib.h>

#define N 32
#define NCHUNK 4

__global__
void thread_multi(int *t1)
{
        int i=blockDim.x * blockIdx.x + threadIdx.x;
        int j=threadIdx.x;
        t1[i]=j;
}

int main()
{
        int *d_t1;
        int *h_t1;
        int i=0;

        cudaStream_t stream[NCHUNK];
        size_t size = N*NCHUNK*sizeof(int);

        cudaMalloc((void **)&d_t1, size);
        cudaMallocHost(&h_t1, size);

        for (i=0;i<N*NCHUNK;i++) {
                h_t1[i]=0;
        }
```

Taken from https://cvw.cac.cornell.edu/gpu/exec_synchronization

```
//Create 4 streams
        for (i = 0; i < NCHUNK;i++) {
                cudaStreamCreate(&stream[i]);
        }

//4 events on each stream - Memory copy to the device, execution, memory copy to the
host, stream destroyed
        for(i=0;i<NCHUNK;i++) {
                cudaMemcpyAsync(d_t1+i*N, h_t1+i*N, N*sizeof(int),
cudaMemcpyHostToDevice, stream[i]);
        }
        for(i=0;i<NCHUNK;i++) {
                cudaStreamSynchronize(stream[i]);
        }

        for(i=0;i<NCHUNK;i++) {
                thread_multi<<<1,N,0,stream[i]>>>(d_t1+i*N);
        }

        for(i=0;i<NCHUNK;i++) {
                cudaStreamSynchronize(stream[i]);
        }

        for(i=0;i<NCHUNK;i++) {
                cudaMemcpyAsync(h_t1+i*N, d_t1+i*N, N*sizeof(int),
cudaMemcpyDeviceToHost, stream[i]);
        }

        for(i=0;i<NCHUNK;i++) {
                cudaStreamSynchronize(stream[i]);
        }
        for (i=0; i < NCHUNK; i++) {
                cudaStreamDestroy(stream[i]);
        }

//Print result
        for(i=0;i<N*NCHUNK;i++) {
                printf("%d: %d\n",i, h_t1[i]);
        }
        cudaFree(d_t1);
        cudaFree(h_t1);
        printf("\nDone\n");
        return 0;
}
```

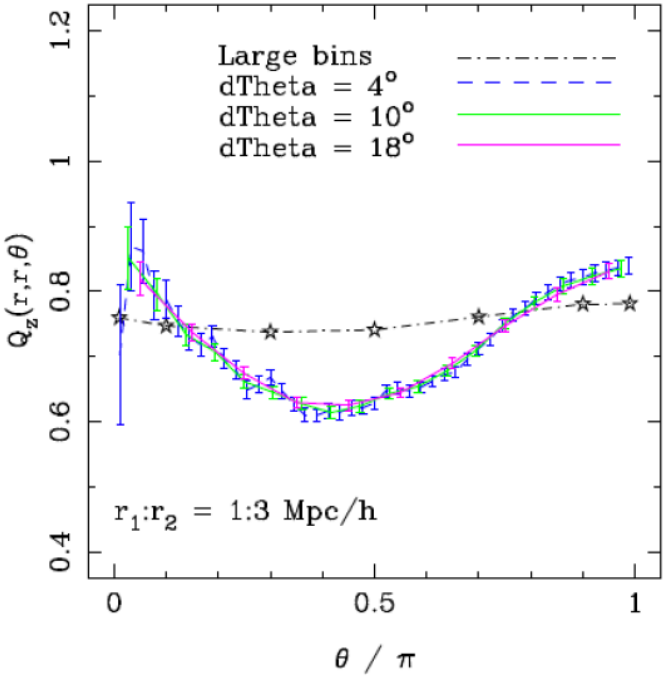
Taken from https://cvw.cac.cornell.edu/gpu/exec_synchronization

Other (advanced) optimisations

- Threads are executed by the GPU controller in sets of 32 (“warp”).
  - Warp will not return until all threads are done with their calculation.
  - “if-then-else”: warp will evaluate each case in turn – if a thread does not hit that case it still has to wait around for the active threads to do their work. -> if/else statements are *serialised*.
- Some instructions are much faster than others
  - E.g. 128 32-bit add, multiply, multiply/add operations per clock cycle, but only 32 log, sin etc operations.
  - http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-instructions__throughput-native-arithmetic-instructions
- Memory coalescing…

3PCF

- Impact of histogram bin size: need to be small enough to resolve "fingers of God" feature
- But too-small bins sizes gives under-sampled bins
- Currently this is the limit of a GPU implementation of the 3PCF: fastest code requires shared memory, but this is limited to 22^3 bins.
- More bins need more memory – global memory slows code by a factor of 2.



Comparison of SDSS projected 3PCF for large/small bin widths (McBride++(2010))

3PCF: where next?

- Do we *have* to use an algorithm that scales as N^3?
  - –k-d trees are not much faster than the brute force calculation.

# galaxies	**KD-Tree**	**CPU full**	**GPU full**
1k	15 s	27 s	4 s
10k	570 s	25776 s	624 s
50k	48089 s	3.2 Gs (extrapolated)	55296 s

- We have significantly more than 50k galaxies in our current datasets!
  - – However will we be able to calculate this for LSST's billions and billions of galaxies?

3PCF via spherical harmonics

- Can measure the 3PCF as:

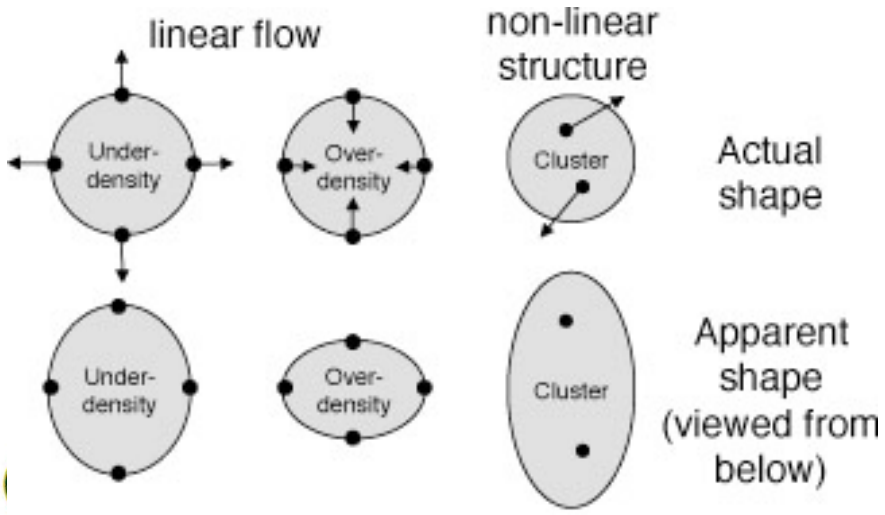
$$\zeta(r_1, r_2; \hat{r}_1 \cdot \hat{r}_2) = \sum_\ell \zeta_\ell(r_1, r_2) P_\ell(\hat{r}_1 \cdot \hat{r}_2)$$

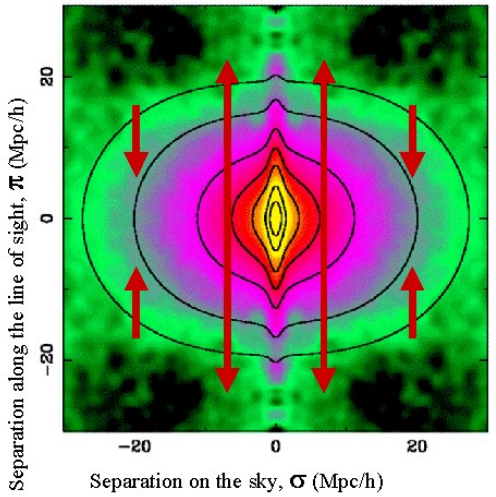
- P_l is a Legendre polynomial. The radial coefficients ζ_l describe the 3PCF dependence on triangle side lengths r_1 and r_2.
  - –Bin density around each galaxy into shells, then expand angular clustering on each shell into spherical harmonics.
  - –Scales as $O(N^2)$!
  - –Basis formed by Legendre polynomials of $r_1.r_2$ is invariant under rotations because dot product is preserved – cannot track anisotropies.

Anisotropic 3PCF

- We actually measure redshifts of galaxies.
  - But, the redshift we measure is a combination of cosmological redshift due to the expansion of the Universe, and galaxy peculiar velocity in their neighbourhood due to local gravitational gradient.

  --> *redshift space distortions probe growth of structure due to gravity*.





Summary

- Programing in CUDA is pretty simple but not always a performance win
  - Try to saturate both the GPU and the memory bus!
- It takes some serious effort to get the last 20% of speed-up, but you learn a lot about your hardware in the process
- HPC is vital for cosmology in the survey era.
- http://docs.nvidia.com/cuda/cuda-c-programming-guide/ is very useful