

# Is Hybrid Programming a Bad Idea who's time has come?

**Vijay Saraswat, IBM TJ Watson**

**The Great Buddha was asked**

**Does God exist?**

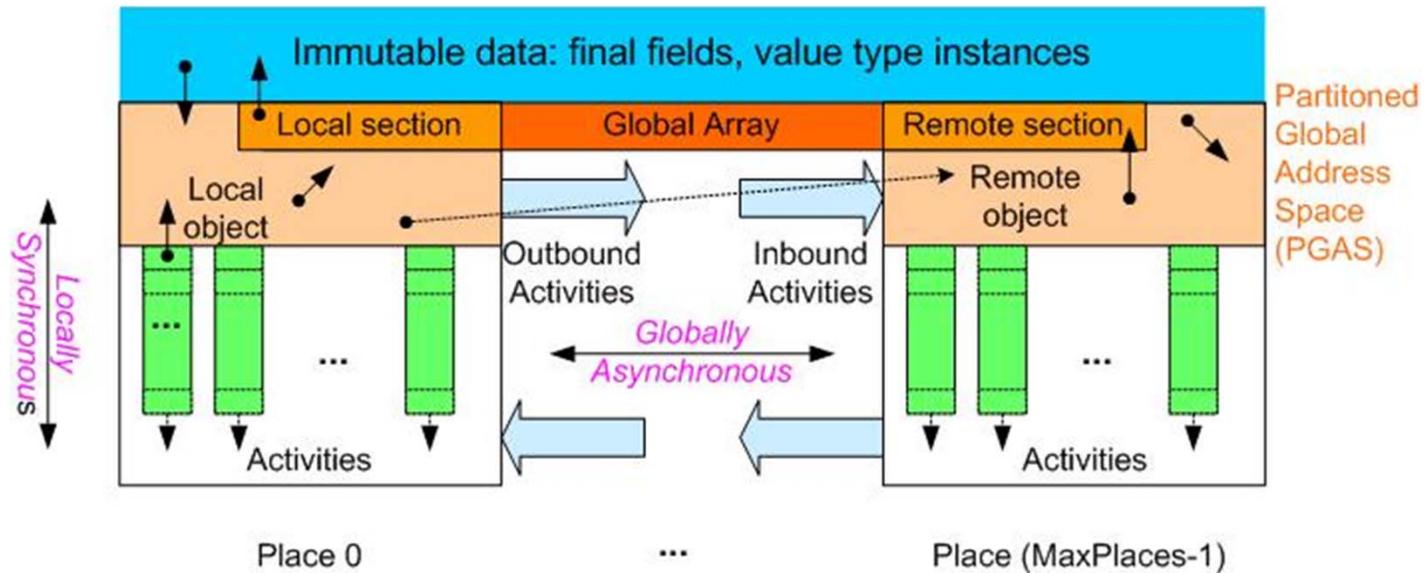
**It doesn't really matter!**

***There are fortunes, reputations to be made!***

***Great technical work to be done!***

***Lets get going!***

# What is Asynchronous PGAS?



<b>Fine grained concurrency</b> • <b>async S</b>	<b>Atomicity</b> • <b>atomic S</b> • <b>when (c) S</b>	<b>Global data-structures</b> • <b>points, regions, distributions, arrays</b>
<b>Place-shifting operations</b> • <b>at (P) S</b>	<b>Ordering</b> • <b>finish S</b> • <b>clocks</b>	<b>Supports</b> • <b>clocked final variables</b> • <b>(clocked) acc</b>

**Two basic ideas: Places and Asynchrony**

# Why Program GPUs with X10?

Why program the GPU at all?

**Many times faster for certain classes of applications**

**Hardware is cheap and widely available**

**Take load off of CPU**

Why X10 (instead of CUDA/OpenCL)?

**For the GPU parts:**

**Easier to program for systems with several GPUs**

**Higher-level abstractions result in fewer lines of kernel code.**

**Same code will also run on a CPU with reasonable performance.**

**Can choose at runtime whether code runs on CPU or GPU.**

**For the CPU parts: the usual reasons for using X10:**

**Simpler programs**

**Easier to write parallel programs**

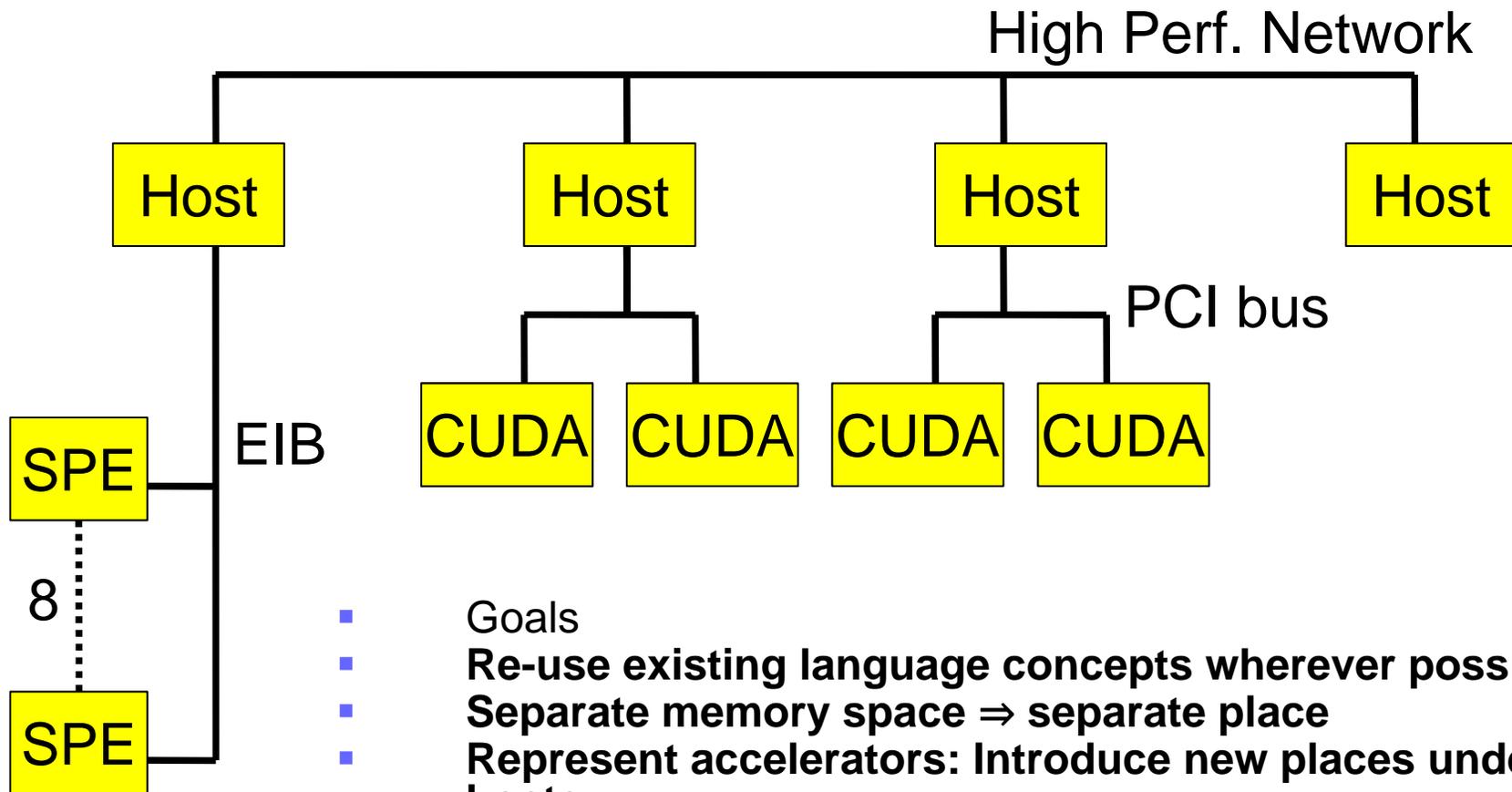
**Much easier to write distributed programs**

**Greater static safety (true for GPU code too)**

Above argument applies for other accelerators like Cell SPEs too!

**X10/CUDA Slides courtesy of David Cunningham, X10 group**

# Concepts of Heterogeneous X10



- Goals
- Re-use existing language concepts wherever possible
- Separate memory space  $\Rightarrow$  separate place
- Represent accelerators: Introduce new places under hosts
- Place count and topology unknown until run-time
- Same X10 code works on different configurations
- Support CUDA now, Cell/OpenCL/etc future work

# The Programmer Experience

GPU programming will never be trivial. We try to make it as easy as possible.

## Getting correctness

**Can debug X10 kernel code on CPU first, using standard techniques**

**Static errors avoid certain classes of faults.**

**Goal: Eliminate all GPU segfaults with no performance limitations.**

Currently detect all dereferences of non-local data

TODO: Static array bounds checking (general X10 goal)

TODO: Static null-pointer checking (general X10 goal)

**Stock 'cudagdb' should work but of limited use on desktop machines.**

**TODO: throwing exceptions on GPU**

**TODO: limited printf debugging on GPU (challenges with string concatenation)**

**Getting performance:** Need understanding of the CUDA performance model

**Must know advantages+limitations of [registers, SHM, global memory]**

**Avoid warp divergence**

← **TODO: high-level language features to abstract from such architectural details**

**Avoid irregular memory access**

**Avoid misaligned memory access**

**Use CUDA profiling tool to debug kernel performance (very easy and usable)**

**Can inspect and disassemble generated cubin file**

**Easier to tune blocks/threads using auto-configuration**

## X10/CUDA code (mass sqrt example)

```

for (host in Place.places) at (host) {
  val init = ValRail.make(1000, (i:Int)=>i as Float);
  val recv = Rail.make(1000, (i:Int)=>0.0 as Float);
  for (p in here.children()) if (p.isCUDA()) {
    val remote = Rail.makeRemote(p, 1000, (Int)=>0.0 as Float);
    val blocks = 8, threads = 64;
    finish async (p) @CUDA {
      for ((block) in 0..blocks-1) {
        for ((thread) in 0..threads-1) async {
          val tid = block*threads + thread;
          val tids = blocks*threads;
          for (var i:Int=tid ; i<1000 ; i+=tids) {
            remote(i) = Math.sqrt(init(i));
          } } } }
      // Console.OUT.println(remote(42));
    finish recv.copyFrom(0, remote, 0, len);
  }
}

```

Discover GPUs

Alloc on GPU

Implicit capture  
and transfer of  
immutable state

kernel

Would be static type error

Copy results back to host

# Kernel Structure

```

finish async (p) @CUDA {
  for ((block) in 0..blocks-1) {
    for ((thread) in 0..threads-1) async {
      va = block*threads + thread;
      va s = blocks*threads;
      for (var i:Int=tid ; i<len ; i+=tids) {
        note(i) = Math.sqrt(init(i));
      }
    }
  }
}
    
```

Real Kernel

Enforces extra restrictions on block

Define kernel shape

Only sequential code

Only primitive types / rails

No allocation, no dynamic dispatch

The future is in new (commercial) applications

*And high level programming models for masses  
of programmers...*

*...Map Reduce*

*...Matlab*

*... R*

*That transparently exploit underlying hardware*

**Analytics slides courtesy of Shiva Vaithyanathan, IBM Research**

# Trend: New "Big Data" becoming commonplace



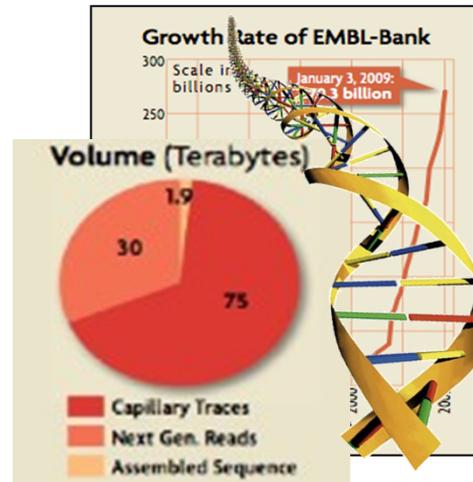
10 Terabytes per day



7 Terabytes per day



Transactions: 46 Terabytes per year



Genomes: Petabytes per year



100 Terabytes per year



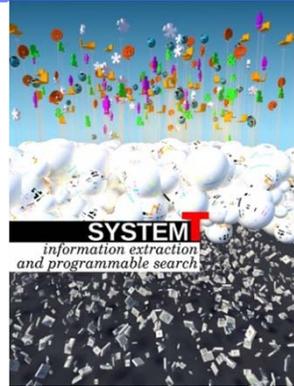
Call Records: 3 Terabytes per day



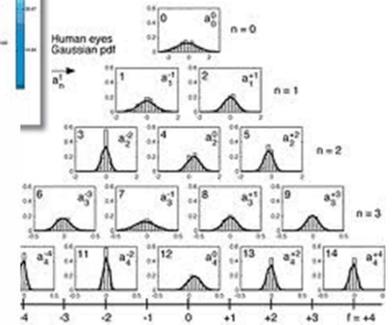
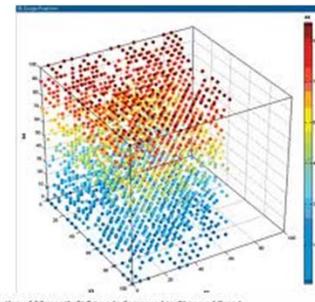
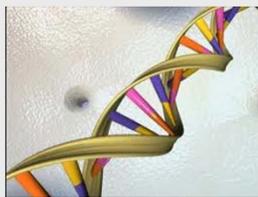
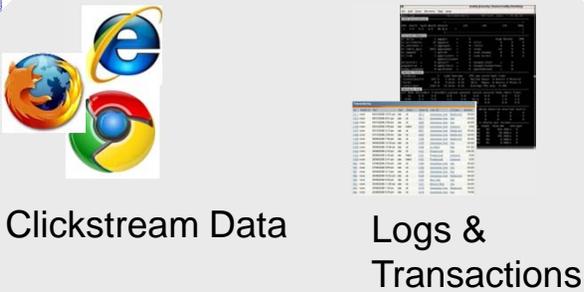
Blogs: 10 Terabytes per year



# It is really "Big Data" AND "New Analytics"



## Text Analytics



## Statistical Model Building