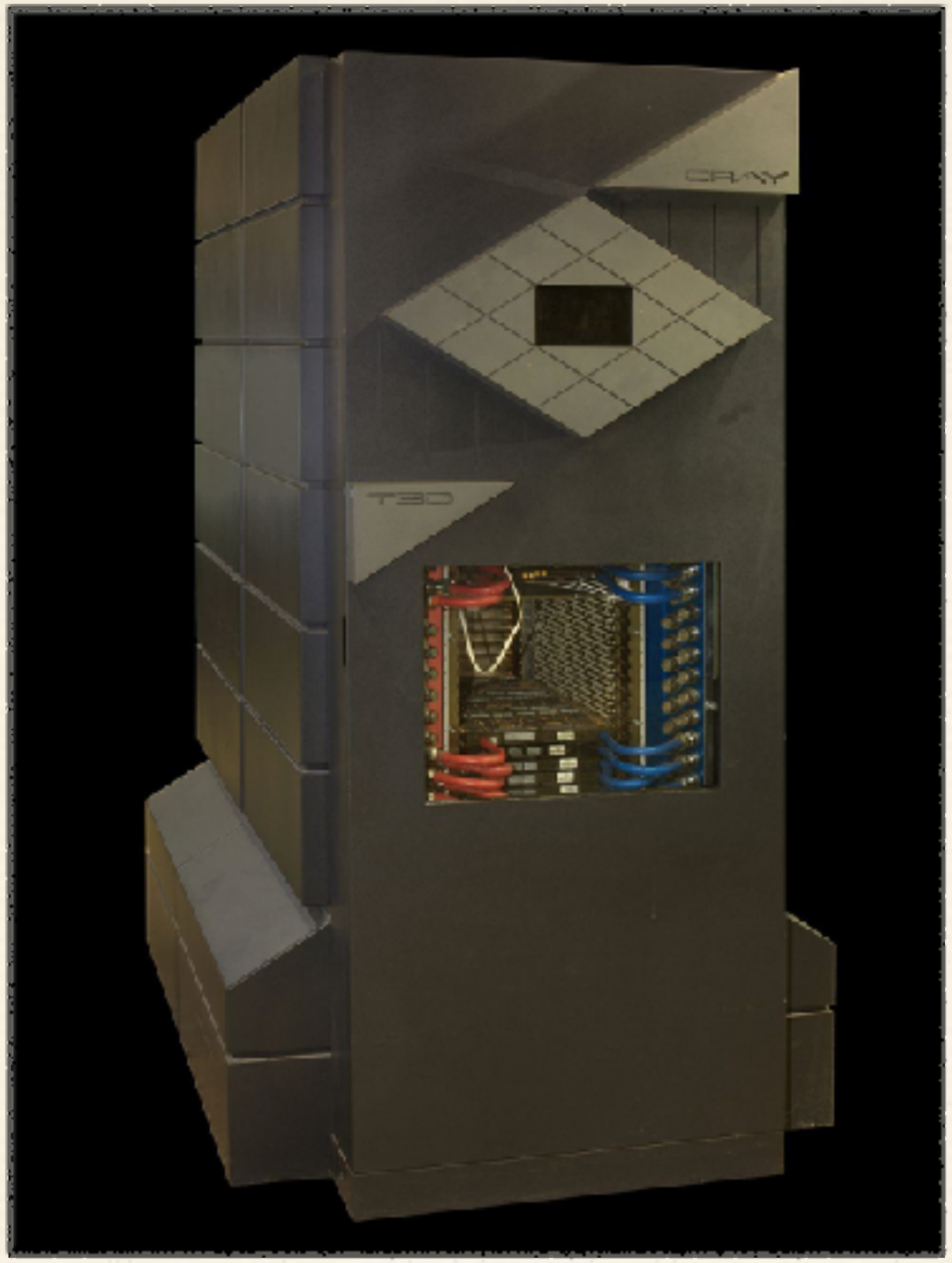
Shared Memory HPC Programming: Past, Present, and Future

Bill Carlson
IDA Center for Computing Sciences
November 13, 2017

Disclaimer

This presentation reflects the personal views of the author.

These views may or may not be held by the author's employer or its sponsors.



Our Problem in 1993

How do we program this? And get good performance?

AC for the Cray T3D

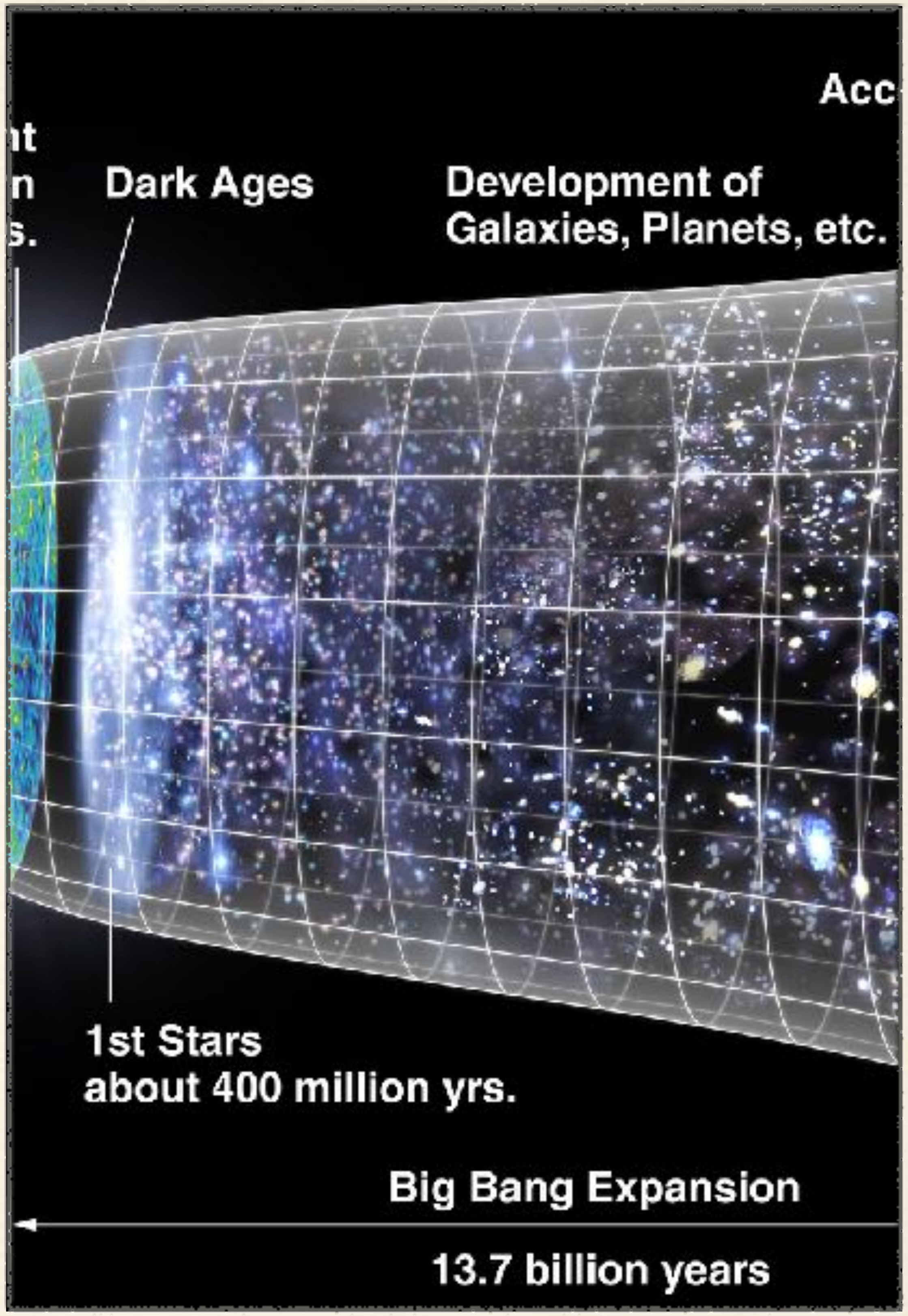
- An outgrowth of our work on CM5
- Shared memory on a distributed memory machine
    - “dist” keyword is the only syntax change
    - Performance high from special hardware on T3D
    - Faster than “shmem” library, due to low overhead

UPC = AC + Split-C + PCP

- Collaboration with UC Berkeley and LLNL
  - Takes “shared” from AC’s “dist”
  - “strict” and “relaxed” shared memory semantics
  - Split barriers: “notify/wait”
  - Locks
  - Adds several data distributions

PGAS: Expanding the collaboration

- SHMEM Library
- CoArray Fortran
- Global Arrays
- Titanium



DARPA HPCS Program
3 New PGAS Languages!

- Fortress: Implicit Parallelism, Strong Types
  - Cool look: like math in both ASCII and Unicode
  - Effort ended in 2012
- X10: Java-like syntax, asynchrony, locales
  - Going strong but, not much "HPC"
- Chapel: Separate Parallelism and Locality
  - Annual Conference, open source distribution

Post-HPCS PGAS

- Habenaro C and UPC++ (Rice)
- UPC++ (Berkeley)
- CoArray C++ (Cray, EPCC)
- HPX (C++/11,14, LSU, FAU)
- XcalableMP (Tskuba)
- GASPI (Fraunhofer)
- OpenSHMEM
- More every time one looks

HPC at a Crossroads

- Path to Exascale is underway:
  - System complexity increasing
  - Huge increase in OPs
  - Lesser increases in memory and communication performance
- Application development is getting harder, not easier

PGAS at a Crossroads

- Many implementations exist of PGAS techniques, essentially all platforms
- Provide a wealth of programming metaphors
- Performance has been shown to be very good
  - A number of cases which exceed best message passing code
    - Because you have a wider choice of algorithms and synchronization
- Programmer "base" is
  - (somewhat) small, and
  - (somewhat) static

Our Problem in 2017

How do we program this? And get good performance?

Key problem: Random Communication

- Many PGAS apps access shared memory randomly
  - Problematic on modern systems, due to relatively high cost of message initiation
- We have an approach called “exstack/Conveyors”:
  - Generate many more accesses than threads
  - Sort them locally, send in batches
  - Take received messages, process locally
  - Repeat until done
  - Several synchronization styles supported

exstack/Conveyors Analysis

- Performance surprisingly good for most apps
  - Most have plenty of parallelism in their accesses
  - Most tolerate the significant added latency
- Programability is a complete mess
  - Ugly, lengthy, complex looping structures in code
  - Have to packetize work by hand
- We want better, much like our previous "AC" work
  - We have tried to improve this and not found key
  - Aware of related efforts, we don't see key there either
  - Yet!

Thought Questions for Today

- How important is HPC performance?
  - And what does “performance” mean?
- How important is HPC programability?
  - And what does “programability” mean?
- How can we deal with HPC system complexity?
- Should we expand the PGAS horizons beyond HPC?

HPC Performance

- Performance has always been *critical* to HPC
  - By definition. If not, why go to the trouble. What matters:
    - Application results per unit time (e.g., day)
    - Application results per unit cost (e.g., $, Watt)
- Ways to improve application performance
  - Understand performance limiters, then
    - Write new code which gets around limiters
      - Includes both tweaks and new algorithms
    - Use new systems which address limiters
      - Hardware, Compilers, OS, etc, etc.

HPC Programability

- Programability has always been *debated* in HPC
  - By definition: Performance is *critical*. What matters:
    - Application performance achieved per unit cost
- Ways to improve programability
  - Apply more or better programmers
  - Apply more or better tools for programmers
  - Apply different programming techniques
- Debate is always about mix and extent of efforts

“An unwritten program has ZERO performance”

–Me

Key HPC Complexity
Multi-Level Parallelism

- Hardware is becoming increasingly hierarchical
  - Start with SMP “nodes” in distributed machines
  - Add threads within cores within processors
    - GPUs and other accelerators only add to the complexity
- Two distinct issues:
  - What is shared among threads on a “node”? But not globally?
  - What controls the parallel activity on a node?

Multi-Level Parallelism?

- Some programming models urge multi-level
  - SHMEM + pthreads or OpenMP
  - Programmers then write two levels of control flow, one for across nodes, one for on nodes
- UPC supports only local and shared
  - PGAS thread per hardware thread seems about right
  - An extension was made to allow shared allocation on node

Is “HPC” the only PGAS “market”?

- Mostly yes
  - Pointless to “partition” a tiny system
- But maybe not!
  - No widely-useful model for programing SMP processors
    - Most restricted to concurrency (e.g., go)
  - PGAS could provide a path to scalable apps
  - PGAS can be powerful metaphor in progammer education

PGAS Future?

- Stay the Course?
- Another Unification?
- New Approach?

Path Forward One: Keep Pressing

- Our current languages and libraries are good!
- Our current programmers are good!
- We are growing friends all the time
- To Do List:
    - Implement github-scale sharing of PGAS utilities
    - Start work on new application areas
    - Develop curriculum

Path Forward Two: New Unification

- UPC took three smaller, locally used languages
  - And made something better than sum of parts
- Several areas to consider
  - Many C++ based PGAS efforts are underway
  - PGAS Multi-treading and asynchrony
    - Including efforts that don’t know they are PGAS, yet :)
- But gaining branding and adoption is always hard

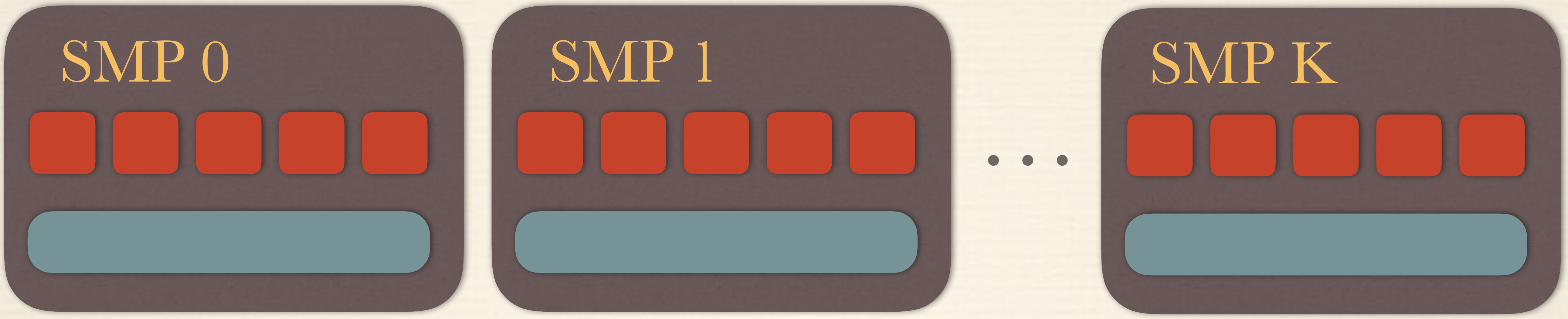
Path Forward Three: New Approach

- What's all this about Python (and friends) then?
  - No need to change the language to change programming models
  - Abundant local computation means we can (maybe) afford the overhead
    - And maybe call "native" code when needed
  - Can us any good communication system
- Why is there no "PGAS" for Python?
  - There is (or was), google "Python PGAS"
  - It was a good thought but never got supported

Consider: JavaScript and Node.js

- Node.js is a JavaScript engine based on Chrome's V8. All I/O is event driven and non-blocking
- Performance: See "Benchmarksgame"
  - JavaScript is about10x Python, 1x go,java, 0.1x C
- Interesting model
  - Every piece of JavaScript is sequential
  - But much parallelism based on event driven "I/O"
  - Which provides good mechanisms to deal with parallelism
- My attempt is to "PGAS" this where performance penalty is acceptable

PGAS Node.js



UPC/PGAS Process: started by SLURM

- Fixed program for inter-node accesses and synchronization

Node.js Process: `fork()/exec()` by UPC/PGAS

- User program, use `mmap()` communication with

PGAS Shared Objects

- All shared data derived from a root shared object
  - Data stored in PGAS program
  - Methods are something like get(), put(), action()
  - Access to shared mediated by PGAS program
  - Completion through callbacks, “promises”
  - May provide the programability and performance

Histogram in JavaScript

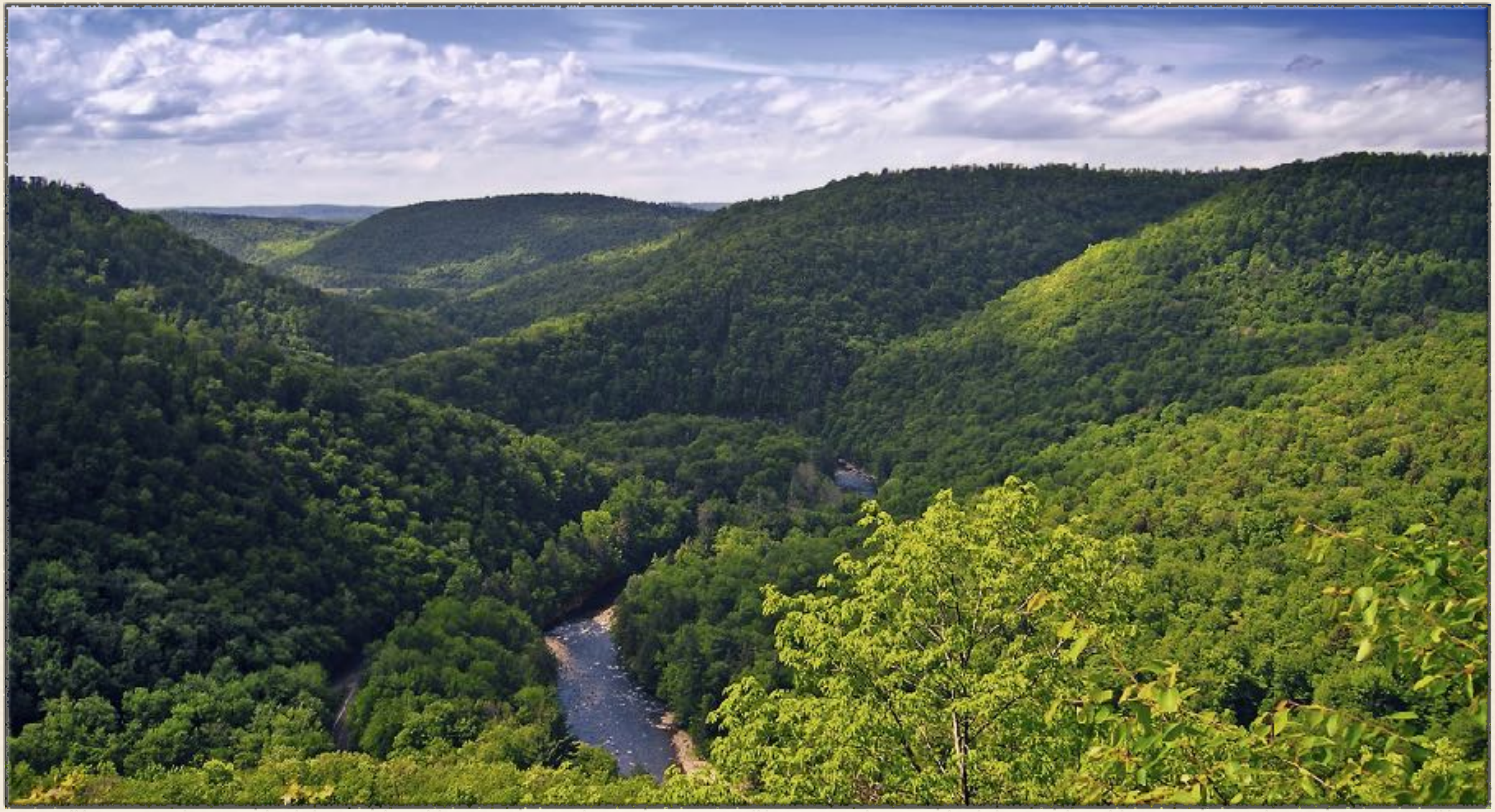
```
let num_per = 2457;
let span = par.num_proc*num_per;

Histogram(par, num_per, function(h) {
 for (let i = 0; i < 12345*par.num_proc; i++) {
  h.update(Math.random()*span));
 }).then(h => print h.histo[0]);
```

- `par` is an object which holds parallel context, from
- `Histogram` is a specialization of `ParallelUpdater`
  - update() parallel communication
  - Callback parallelizes allocation
  - Promise ".then()" handles completion
- Really just starting here, but I'm excited!

Closing Thoughts

- PGAS has developed a significant history
- Currently at a crossroads:
  - Concepts have proved useful in HPC
  - Struggling with current architectures
  - Adapt new approaches
- All future paths are interesting
  - But JavaScript is fun for me now, but happy to help with others!



Future Vistas for PGAS

The fun has only begun

Image Credits

- T3D Image: CC-BY-SA-2.0-fr Rama, Wikimedia: CRAY-T3D IMG 8981-82-87-89.CR2.jpg
- Expanding Image: Public Domain, "CMB Timeline300 no WMAP" by NASA Wikimedia:CMB_Timeline300_no_WMAP.jpg
- Crossroads Image: CC-BY-SA Umberto Nicoletti, flickr.com/photos/unicoletti/2851575552
- Titan Image: Public Domain, James086, Wikimedia: Titan render.png
- Final Image: CC-BY-2.0 Nicholas A. Tonelli, flickr.com/photos/nicholas_t/6697202819

Backup Slides

DARPA's HPCS Program

- High ***Productivity*** Computing Systems
- Productivity: Output per unit of Input
  - Output is problems solved
  - Input is money, energy, people time
- Goal: Increase productivity of HPC by 10x:
  - Systems performance 10x for many metrics
  - Algorithm and Software developers 10x effective in making good code
  - System operators spend 1/10 effort to manage system

New Uses for HPC/PGAS

- A lot of emphasis on "Big Data"
  - How about an awesomely fast PGAS key-value store
- Machine "Deep" "learning"
  - Can PGAS allow real advances in this field
- Previously "Abandoned" HPC applications
  - Industrial uses in manufacturing
- My assertions:
  - PGAS approaches could help add new application areas
  - These are not using HPC (much) because it is too hard to program

Complexity of Next Gen HPC?

- Strong forces for higher complexity
  - Need to control energy leads to specialization
    - Accelerators like GPUs
    - Small, specialized memories
    - Communication at a distance is always limited by cost
  - ExaScale goals are pushing for large performance gains
- Some trends to lower complexity
  - Many applications can fit "on a node" or small segment of system
  - Communication bound algorithms might ignore complex parts of system