Hewlett Packard Enterprise

Reflections on 30 Years of HPC Programming:

Brad Chamberlain

HIPS 2025 Keynote

June 3, 2025

Hewlett Packard Enterprise

Reflections on 30 Years of HPC Programming: So many hardware advances…

Brad Chamberlain

HIPS 2025 Keynote

June 3, 2025

Reflections on 30 Years of HPC Programming: So many hardware advances… So little adoption of new languages

Brad Chamberlain

HIPS 2025 Keynote

June 3, 2025


A Bit About Me

HIPS is a perfect match for my interests

From the workshop overview:

> “The 30th HIPS workshop, to be held as a full-day meeting at the IPDPS 2025 conference in Milan, Lombardy, Italy, **focuses on high-level programming of multiprocessors, compute clusters, and massively parallel machines.** Like previous workshops in this series - established in 1996 - this event serves as a **forum for research in the areas of** parallel applications, **language design, compilers, runtime systems, and programming tools.** It provides a timely forum for scientists and engineers to present the latest ideas and findings in these rapidly changing fields. In our call for papers, **we especially invite papers demonstrating innovative approaches in the areas of emerging programming models for large-scale parallel systems and many-core architectures**. This year we will add to the list topics programming models and environments for the Edge-Cloud-HPC Continuum as well as the application of recent AI technologies in high-level programming models.”

Disclaimer

Lots of personal opinions follow that don't necessarily reflect anyone's views other than mine.
(as with any good keynote)

They may also represent something of a US-oriented perspective?
(I'll be curious if you think so)

HPC: 30 years ago vs. now

Top HPC Systems, June 1995

- Top 5 systems in the Top500
  - **Cores:** 80–3680 cores
  - **Rmax:** ~98.9–170 GFlop/s
  - **Vendors:** Fujitsu, Intel Paragon XP/S, Cray T3D
  - **Networks:** crossbar, mesh, 3D torus

TOP500 LIST - JUNE 1995

R_{max} and R_{peak} values are in GFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Numerical Wind Tunnel, Fujitsu National Aerospace Laboratory of Japan Japan	140	170.00	235.79	
2	XP/S140, Intel Sandia National Laboratories United States	3,680	143.40	184.00	
3	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States	3,072	127.10	154.00	
4	T3D MC1024-8, Cray/HPE Government United States	1,024	100.50	153.60	
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00	

Top HPC Systems, June 2025

- Top 5 systems in the Top500 (results from Nov 2024)
  - **Cores:** 2,073,600–11,039,616 cores (~563x–138,000x)
  - **Rmax:** ~477.9–1742.0 PFlop/s (~2,810,000x–17,600,000x)
  - **Vendors:** HPE/Cray, Microsoft
  - **Networks:** Slingshot-11, InfiniBand NDR

TOP500 LIST - NOVEMBER 2024

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461

What changes did HPC hardware see over that time?

I'd consider these to be the biggest HPC hardware changes over the past 30 years:

- multicore processors
- multi-socket compute nodes
- NUMA memory architectures within compute nodes
- high-radix, low-diameter network interconnects
- GPU computing
- massive-scale HPC systems
- cloud computing

Most of these changes have been driven by striving for better performance and/or energy efficiency

Top HPC Systems, June 1995

- Top 5 systems in the Top500
  - **Cores:** 80–3680 cores
  - **Rmax:** ~98.9–170 GFlop/s
  - **Vendors:** Fujitsu, Intel Paragon XP/S, Cray T3D
  - **Networks:** crossbar, mesh, 3D torus

TOP500 LIST - JUNE 1995

R_{max} and R_{peak} values are in GFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← | 1-100 | 101-200 | 201-300 | 301-400 | 401-500 | →

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Numerical Wind Tunnel, Fujitsu National Aerospace Laboratory of Japan Japan	140	170.00	235.79	
2	XP/S140, Intel Sandia National Laboratories United States	3,680	143.40	184.00	
3	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States	3,072	127.10	154.00	
4	T3D MC1024-8, Cray/HPE Government United States	1,024	100.50	153.60	
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00	

Adopted HPC Programming Notations, June 1995

- Top 5 systems in the Top500
  - **Cores:** 80–3680 cores
  - **Rmax:** ~98.9–170 GFlop/s
  - **Vendors:** Fujitsu, Intel Paragon XP/S, Cray T3D
  - **Networks:** crossbar, mesh, 3D torus

- HPC Programming Notations:
  - **Languages:** C, C++, Fortran
  - **Inter-node:** MPI, SHMEM
  - **intra-node:** vendor-specific pragmas and intrinsics
    - OpenMP on the horizon: 1997

TOP500 LIST - JUNE 1995

R_{max} and R_{peak} values are in GFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Numerical Wind Tunnel, Fujitsu National Aerospace Laboratory of Japan Japan	140	170.00	235.79	
2	XP/S140, Intel Sandia National Laboratories United States	3,680	143.40	184.00	
3	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States	3,072	127.10	154.00	
4	T3D MC1024-8, Cray/HPE Government United States	1,024	100.50	153.60	
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00	

Adopted HPC Programming Notations, June 2025

- Top 5 systems in the Top500 (results from Nov 2024)
  - **Cores:** 2,073,600–11,039,616 cores (~563x–138,000x)
  - **Rmax:** ~477.9–1742.0 PFlop/s (~2,810,000x–17,600,000x)
  - **Vendors:** HPE/Cray, Microsoft
  - **Networks:** Slingshot-11, InfiniBand NDR

- HPC Programming Notations:
  - **Languages:** C, C++, Fortran
  - **Inter-node:** MPI, SHMEM, Fortran 2008 coarrays
  - **Intra-node:** OpenMP
  - **GPUs:** CUDA, HIP, OpenMP, OpenCL, OpenACC, Kokkos, SYCL, …

TOP500 LIST - NOVEMBER 2024

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

← 1-100 101-200 201-300 301-400 401-500 →

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461

Adopted HPC Programming Notations, June 1995 vs. June 2025

June 1995:

- **Languages:** C, C++, Fortran
- **Inter-node:** MPI, SHMEM
- **intra-node:** vendor-specific pragmas and intrinsics

June 2025:

- **Languages:** C, C++, Fortran
- **Inter-node:** MPI, SHMEM, Fortran 2008 coarrays
- **Intra-node:** OpenMP
- **GPUs:** CUDA, HIP, OpenMP, OpenCL, OpenACC, Kokkos, SYCL, …

Despite 30 years of amazing HPC progress in performance and efficiency, we have not broadly adopted any new HPC programming languages

HPC programming may have even lost ground due to hardware trends

Losing Ground as HPC Hardware Changes?

- Most of the recent hardware changes we've made have hurt, rather than helped, HPC programmability
  - multicore processors **hurt**
  - multi-socket compute nodes **hurt**
  - NUMA memory architectures within compute nodes **hurt**
  - high-radix, low-diameter network interconnects **helped**
  - GPU computing **hurt**
  - massive-scale HPC systems **neutral**
  - cloud computing **neutral**

- Our notations haven't been sufficiently rich and general-purpose for parallelism and locality
  - If they had, we wouldn't write programs that needed C++ *and* MPI *and* OpenMP *and/or* CUDA

Do we need HPC languages?

- Libraries, directives, and extensions have obviously gotten us quite far
  - Virtually all notable HPC computations from the past 30 years have used them

- That said, most HPC programming is still quite low-level and mechanism-oriented
  - e.g., “send this message”, “spawn these threads”, “launch this kernel”
  - These are important capabilities for control over performance
  - Yet they need not be our stopping point

- We’re living in a state similar to Fortran’s introduction—skeptical about higher-level approaches
  - Just as Fortran did not remove the ability to write assembly, HPC languages should support manual overrides

Why Consider New Languages at all?

Syntax	• High level, elegant syntax • Improve programmer productivity
Semantics	• Static analysis can help with correctness • We need a compiler (front-end)
Performance	• If optimizations are needed to get performance • We need a compiler (back-end)
Algorithms	• Language defines what is easy and hard • Influences algorithmic thinking

[Source: Kathy Yelick, CHIUW 2018 keynote: *Why Languages Matter More Than Ever*, used with permission]

Is the lack of newly-adopted HPC languages due to lack of trying?

Notable HPC Programming Languages of the past 30 years

Mid-to-late 90's Classics:

- HPF: High Performance Fortran
- ZPL
- NESL

PGAS founding members:

- CAF: Coarray Fortran
- UPC
- Titanium

C-based approaches:

- Cilk
- SAC: Single-Assignment C

HPCS-era languages:

- Chapel
- Fortress
- X10
- CAF 2.0

Post-HPCS:

- XcalableMP
- Regent

Embedded pseudo-languages (a slippery slope!)

- Charm++, Global Arrays, HPX, UPC++, Legion, …

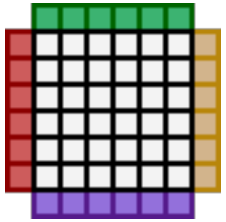
I don't mean to imply that all these languages were worthy of success (nor even most of them)

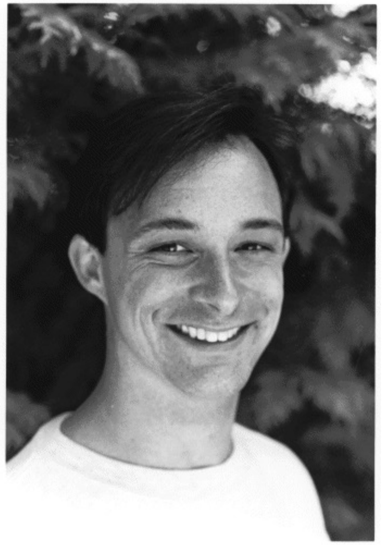
Let's look at one that definitely wasn't…

ZPL

Me, 30 years ago

- Masters-level grad student at University of Washington, working on ZPL





ZPL:

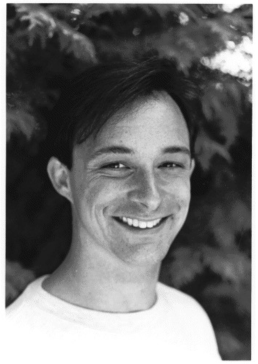
- a data-parallel array language designed for HPC
  - parallelism expressed through first-class index sets called *regions*
- supported a *WYSIWYG performance model*
  - syntax indicated presence, and style, of communication
  - published at HIPS 1998

“Programming Language Design Ceased to be Relevant in the 1980s.”

-Anonymous reviewer on a rejected ZPL paper, circa 1995
(paraphrased, from memory)

Why this review didn't derail my career

[setting: The office of Larry Snyder, my PhD advisor, after we received the review]



Me (demoralized):

- "Why are we even bothering to throw ourselves at this wall, given attitudes like these?"
- "Let's just use our HPC smarts to go and solve some big, cool science problem!"

Larry:

- "If we solve a cool science problem, then we've solved one problem, whereas…"
- "If we create a great language, we magnify our effort by helping others solve their cool problems."

This conversation is a huge part of why I've essentially spent my career devoted to this topic

In retrospect, think about how wrong that attitude was

Consider all the currently relevant languages that emerged, or rose to prominence, during those 30 years:

- **Java** (~1995)
- **Javascript** (~1995)
- **Python** (~1989; v2.0 ~2000)
- **C#** (~2000)
- **Go** (~2009)
- **Rust** (~2012)
- **Julia** (~2012)
- **Swift** (~2014)
- …

These languages have become favorite day-to-day languages of many users across multiple disciplines

What made these languages "stick"?

What focus areas distinguished these languages and helped them take hold? In my opinion…

• **Java** (~1995)	safety, portability, OOP, www
• **Javascript** (~1995)	productivity, www
• **Python** (~1989; v2.0 ~2000)	productivity, extensibility
• **C#** (~2000)	safety, productivity, OOP
• **Go** (~2009)	concurrency, productivity
• **Rust** (~2012)	safety, performance
• **Julia** (~2012)	productivity, interoperability, library re-use, performance
• **Swift** (~2014)	productivity, safety
• …	

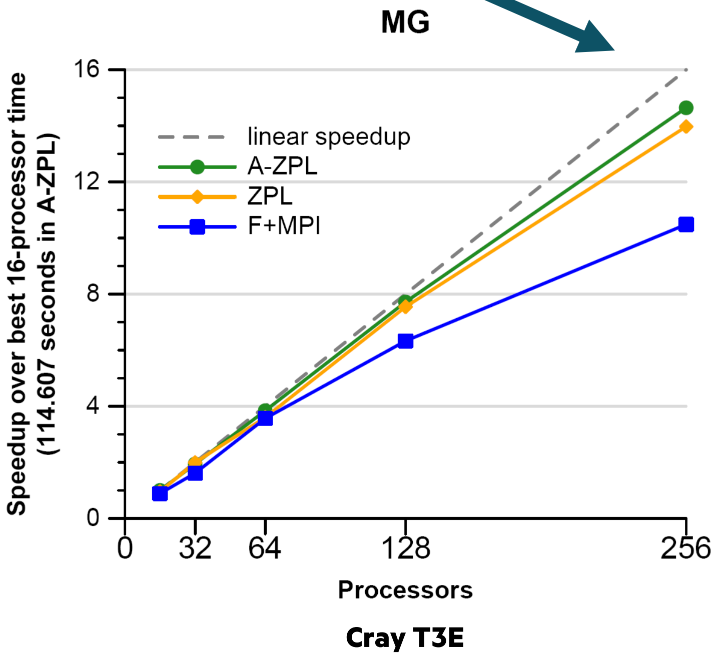
Frequent themes: productivity, safety, portability, performance (things we also value in HPC!)

Parallelism or concurrency? Typically supported, but rarely a primary theme

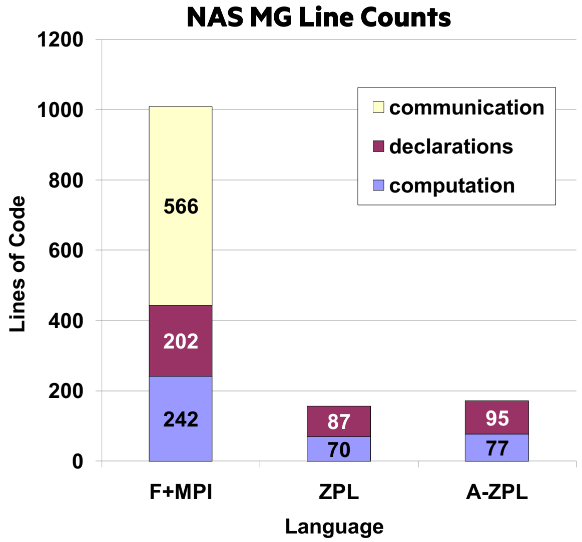
Support for locality control or scalability? Virtually none

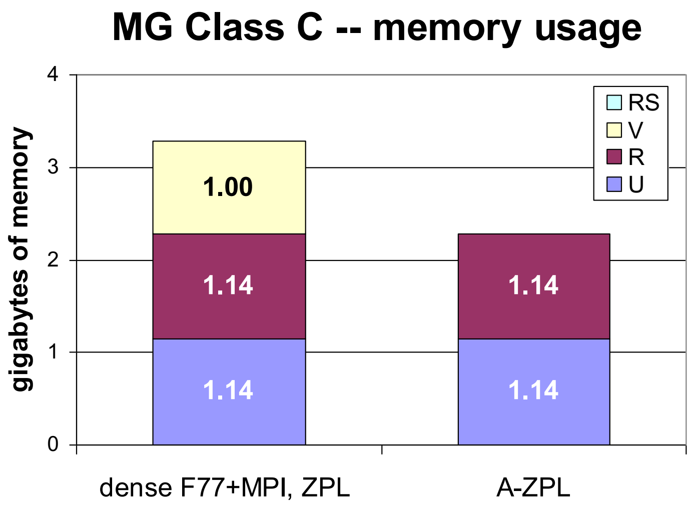
Sample ZPL Result circa 2001: NAS MG (ZPL vs. Fortran+MPI)

ZPL could outscale the reference MPI version, using less code (and clear code at that), and less memory




```
procedure rprj3(var S,R: [,,] double;
                    d: array [] of direction);
begin
  S := 0.5000 * R +
       0.2500 * (R@^d[ 1, 0, 0] + R@^d[ 0, 1, 0] + R@^d[ 0, 0, 1] +
                 R@^d[-1, 0, 0] + R@^d[ 0,-1, 0] + R@^d[ 0, 0,-1] +
       0.1250 * (R@^d[ 1, 1, 0] + R@^d[ 1, 0, 1] + R@^d[ 0, 1, 1] +
                 R@^d[ 1,-1, 0] + R@^d[ 1, 0,-1] + R@^d[ 0, 1,-1] +
                 R@^d[-1, 1, 0] + R@^d[-1, 0, 1] + R@^d[ 0,-1, 1] +
                 R@^d[-1,-1, 0] + R@^d[-1, 0,-1] + R@^d[ 0,-1,-1])+
       0.0625 * (R@^d[ 1, 1, 1] + R@^d[ 1, 1,-1] +
                 R@^d[ 1,-1, 1] + R@^d[ 1,-1,-1] +
                 R@^d[-1, 1, 1] + R@^d[-1, 1,-1] +
                 R@^d[-1,-1, 1] + R@^d[-1,-1,-1]);
end;
```





Why wasn't ZPL broadly adopted? Why was that appropriate?

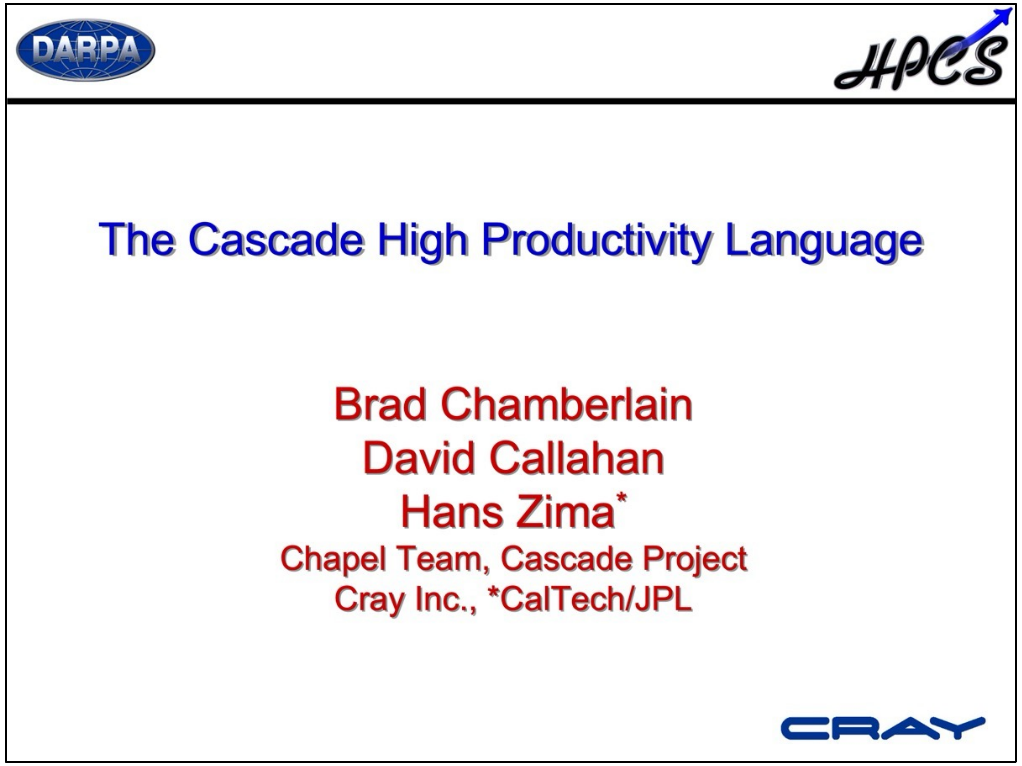
- ZPL was a great academic language
  - Chose the thing we wanted to study, and studied it well
    - Specifically, scalable, array-based data parallelism with syntactically visible communication

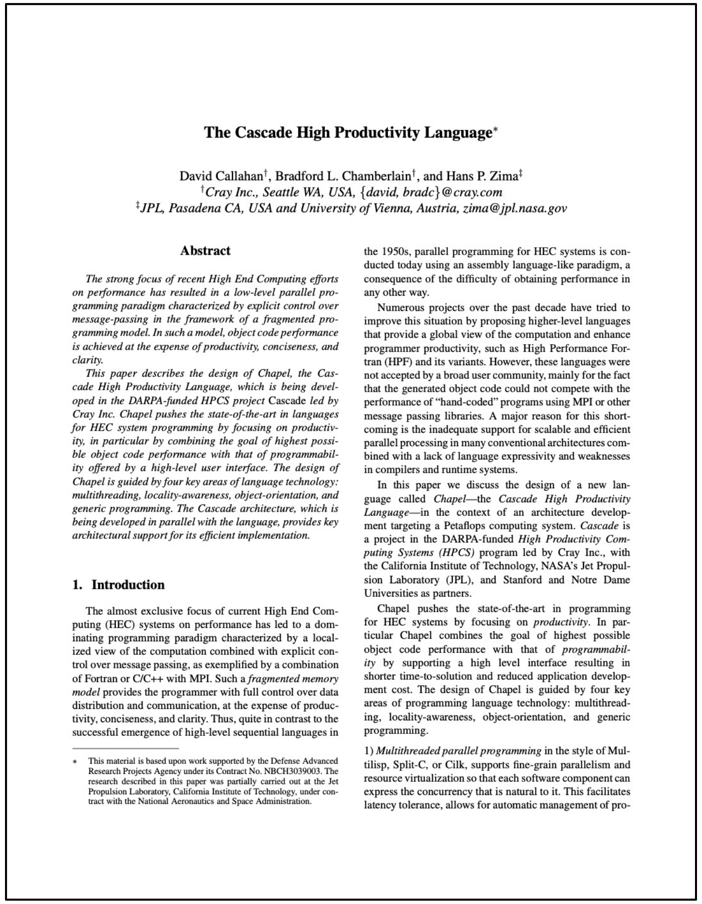
- Yet, it was not a very practical one
  - Supporting only one level of data-parallelism is too restrictive for many real scientific computations
    - It also would've turned out to be insufficient for GPU computing
  - Didn't support features practical users would want: OOP, generic programming, interoperability, modularity, …

- Like so many other HPC notations, insufficiently rich support for expressing parallelism & locality

So, ZPL failed… Do we give up?

- Not at all! It is crucial to learn from failure and improve
  - We learned from ZPL, and also from the failures and struggles of others: HPF, NESL, Sisal, Cilk, UPC, CAF, …
  - And from that, came the **C**ascade **H**igh **P**roductivity **L**anguage, Chapel!
    - original Chapel paper published at HIPS 2004





The Cascade High Productivity Language*

David Callahan†, Bradford L. Chamberlain†, and Hans P. Zima‡
†*Cray Inc., Seattle WA, USA, {david, bradc}@cray.com*
‡*JPL, Pasadena CA, USA and University of Vienna, Austria, zima@jpl.nasa.gov*

Abstract

The strong focus of recent High End Computing efforts on performance has resulted in a low-level parallel programming paradigm characterized by explicit control over message-passing in the framework of a fragmented programming model. In such a model, object code performance is achieved at the expense of productivity, conciseness, and clarity.

This paper describes the design of Chapel, the Cascade High Productivity Language, which is being developed in the DARPA-funded HPCS project Cascade *led by Cray Inc. Chapel pushes the state-of-the-art in languages for HEC system programming by focusing on productivity, in particular by combining the goal of highest possible object code performance with that of programmability offered by a high-level user interface. The design of Chapel is guided by four key areas of language technology: multithreading, locality-awareness, object-orientation, and generic programming. The Cascade architecture, which is being developed in parallel with the language, provides key architectural support for its efficient implementation.*

1. Introduction

The almost exclusive focus of current High End Computing (HEC) systems on performance has led to a dominating programming paradigm characterized by a localized view of the computation combined with explicit control over message passing, as exemplified by a combination of Fortran or C/C++ with MPI. Such a *fragmented memory model* provides the programmer with full control over data distribution and communication, at the expense of productivity, conciseness, and clarity. Thus, quite in contrast to the successful emergence of high-level sequential languages in the 1950s, parallel programming for HEC systems is conducted today using an assembly language-like paradigm, a consequence of the difficulty of obtaining performance in any other way.

Numerous projects over the past decade have tried to improve this situation by proposing higher-level languages that provide a global view of the computation and enhance programmer productivity, such as High Performance Fortran (HPF) and its variants. However, these languages were not accepted by a broad user community, mainly for the fact that the generated object code could not compete with the performance of "hand-coded" programs using MPI or other message passing libraries. A major reason for this shortcoming is the inadequate support for scalable and efficient parallel processing in many conventional architectures combined with a lack of language expressivity and weaknesses in compilers and runtime systems.

In this paper we discuss the design of a new language called *Chapel*—the *Cascade High Productivity Language*—in the context of an architecture development targeting a Petaflops computing system. *Cascade* is a project in the DARPA-funded *High Productivity Computing Systems (HPCS)* program led by Cray Inc., with the California Institute of Technology, NASA's Jet Propulsion Laboratory (JPL), and Stanford and Notre Dame Universities as partners.

Chapel pushes the state-of-the-art in programming for HEC systems by focusing on *productivity*. In particular Chapel combines the goal of highest possible object code performance with that of *programmability* by supporting a high level interface resulting in shorter time-to-solution and reduced application development cost. The design of Chapel is guided by four key areas of programming language technology: multithreading, locality-awareness, object-orientation, and generic programming.

1) *Multithreaded parallel programming* in the style of Multilisp, Split-C, or Cilk, supports fine-grain parallelism and resource virtualization so that each software component can express the concurrency that is natural to it. This facilitates latency tolerance, allows for automatic management of pro-

* This material is based upon work supported by the Defense Advanced Research Projects Agency under its Contract No. NBCH3039003. The research described in this paper was partially carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Chapel

What is Chapel?

Chapel: A modern parallel programming language

- Portable & scalable
- Open-source & collaborative

Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive

Productive Parallel Programming: One Definition

Imagine a programming language for parallel computing that is as…

…**readable and writeable** as Python

…yet also as…

…**fast** as Fortran / C / C++

…**scalable** as MPI / SHMEM

…**GPU-ready** as CUDA / HIP / OpenMP / Kokkos / OpenCL / OpenACC / …

…**portable** as C

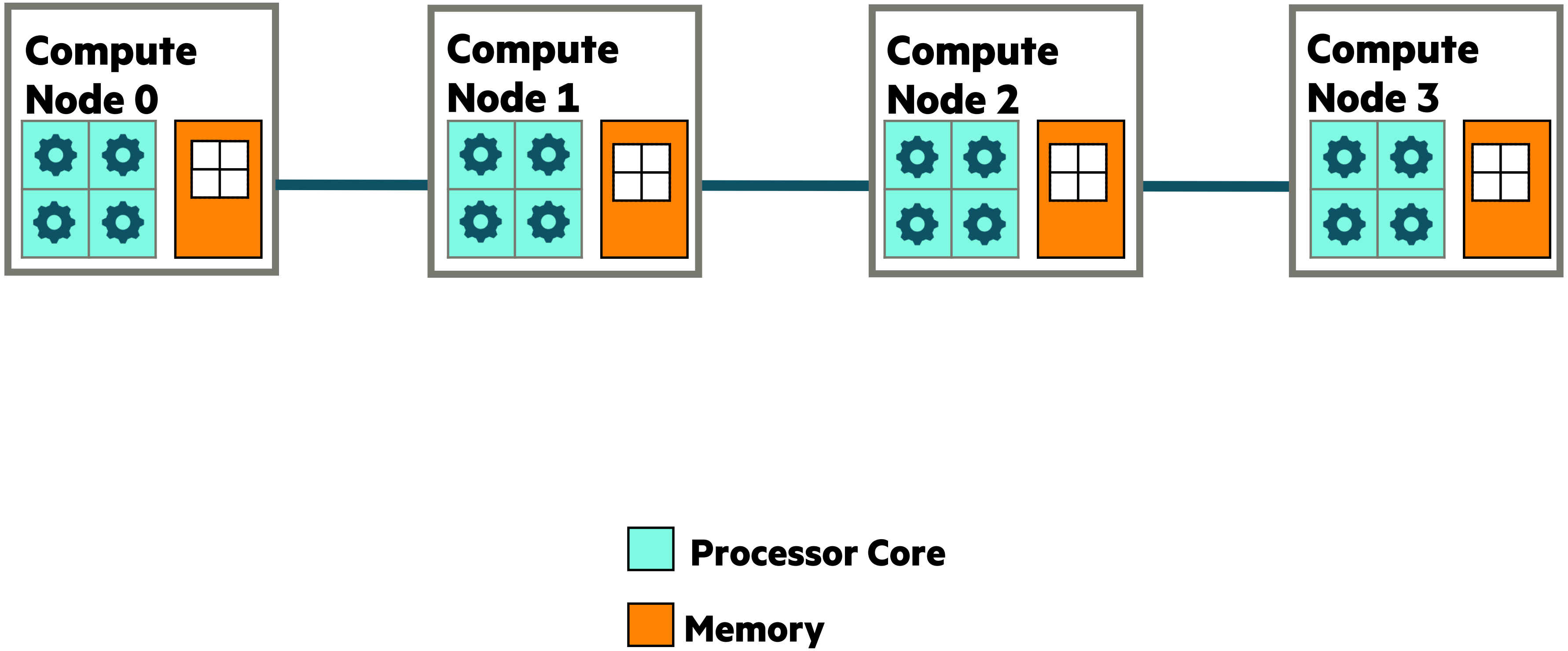
…**fun** as [your favorite programming language]

This is our motivation for Chapel

Chapel Features for Parallelism and Locality, Briefly

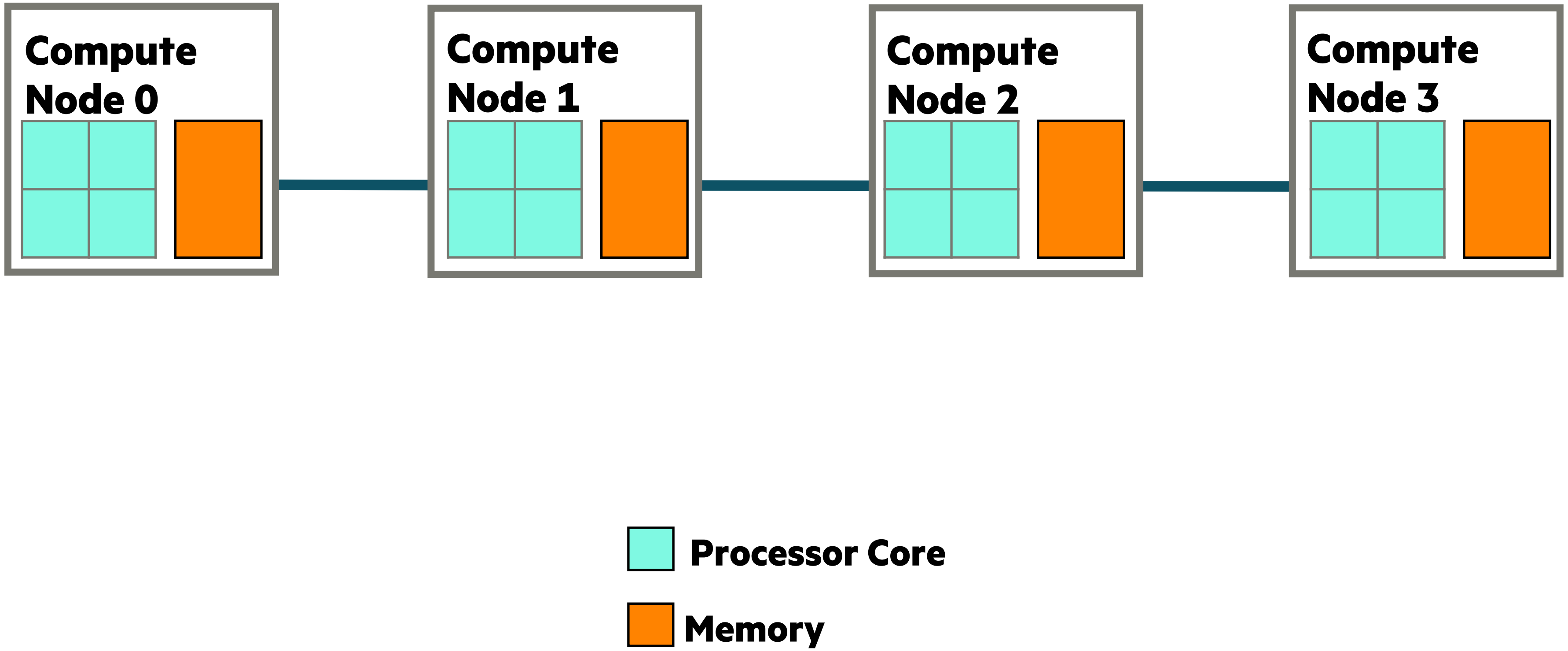
Key Concerns for Scalable Parallel Computing

1. **parallelism:** What tasks should run simultaneously?
2. **locality:** Where should the tasks run? Where should data be allocated?



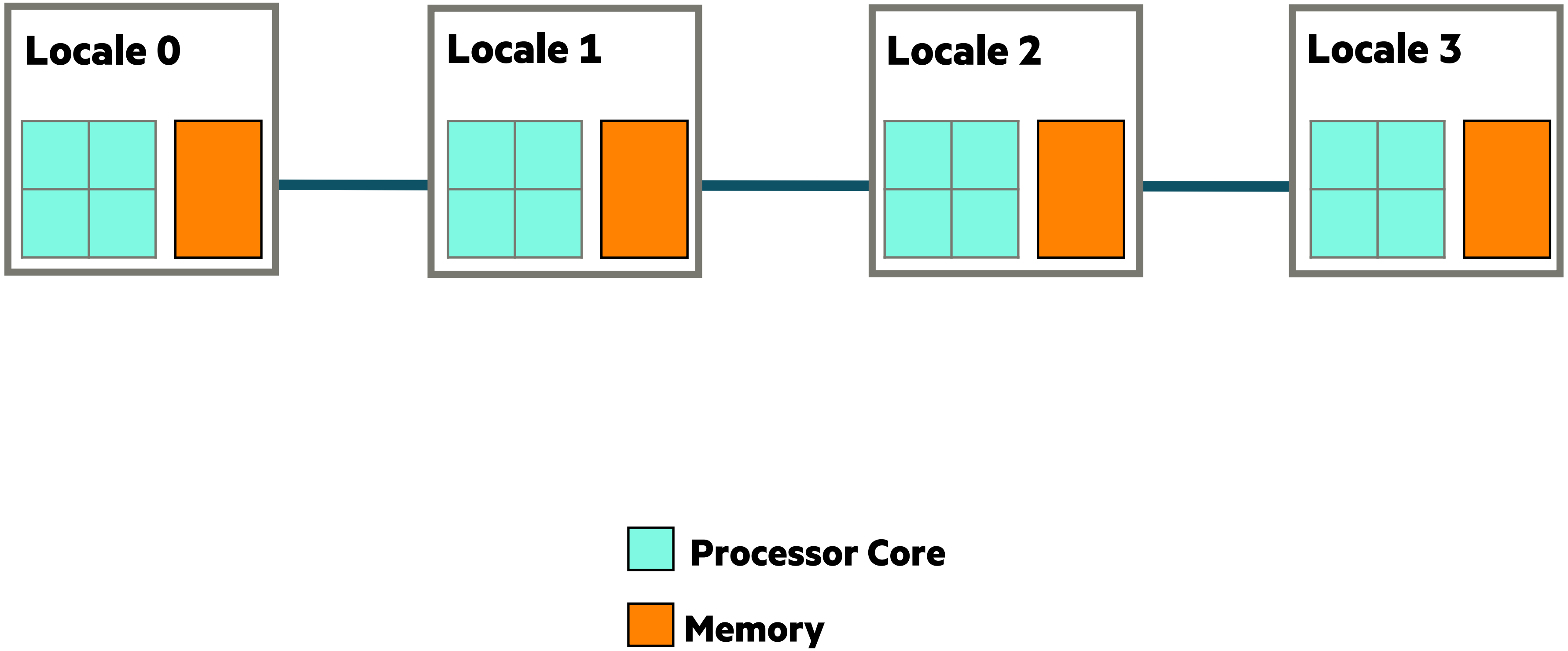
Locales in Chapel

- In Chapel, a *locale* refers to a compute resource with…
  - processors, so it can run tasks
  - memory, so it can store variables
- For now, think of each compute node as being a locale



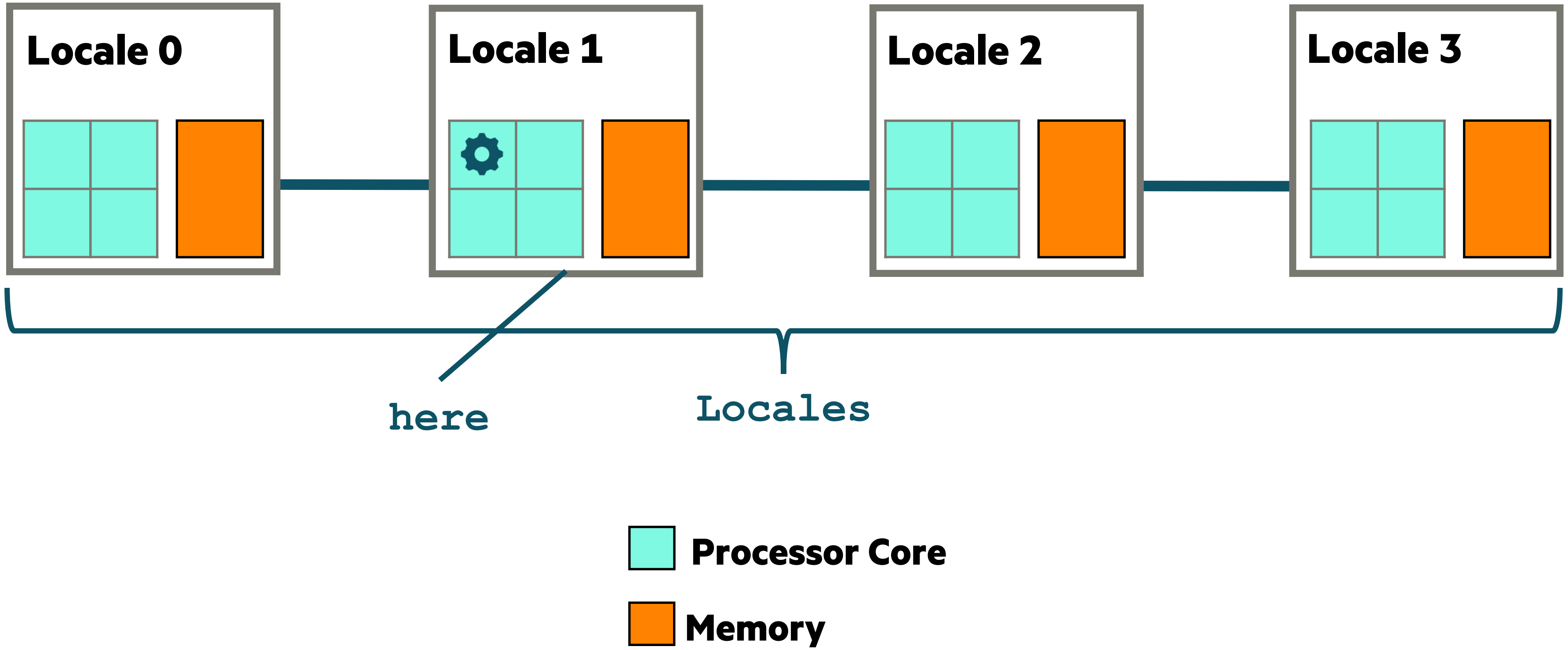
Locales in Chapel

- In Chapel, a *locale* refers to a compute resource with…
  - processors, so it can run tasks
  - memory, so it can store variables
- For now, think of each compute node as being a locale



Built-In Locale Variables in Chapel

- Two key built-in variables for referring to locales in Chapel programs:
  - `Locales`: An array of locale values representing the system resources on which the program is running
  - `here`: The locale on which the current task is executing



Basic Features for Locality

basics-on.chpl

```
writeln("Hello from locale ", here.id);

var A: [1..2, 1..2] real;

for loc in Locales {
  on loc {
    var B = A;
  }
}
```

All Chapel programs begin running as a single task on locale 0

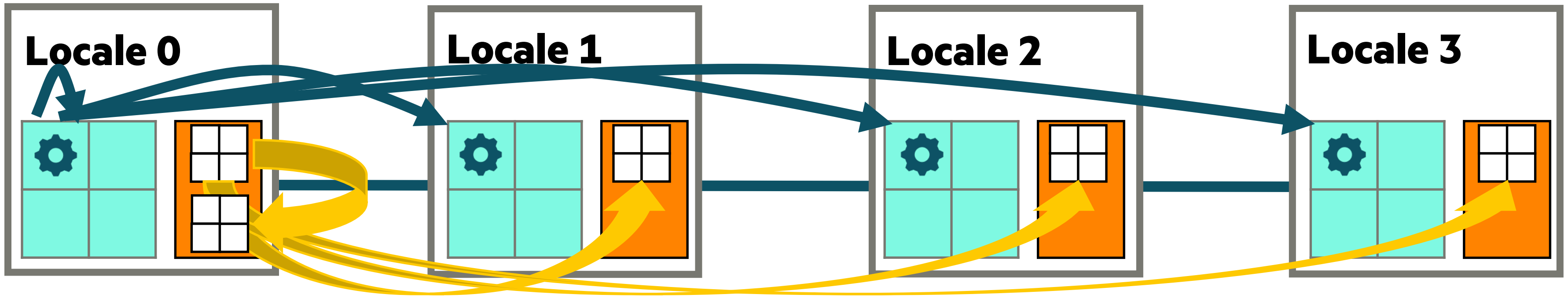
Variables are stored using the memory local to the current task

This loop will serially iterate over the program's locales

on-clauses move tasks to target locales

remote variables can be accessed directly

This is a distributed, yet serial, computation



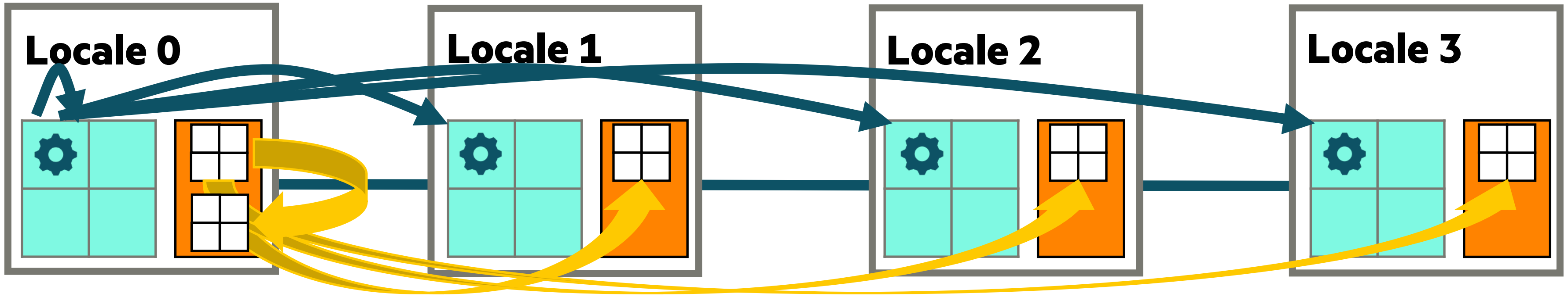
Mixing Locality with Task Parallelism

basics-coforall.chpl

```
writeln("Hello from locale ", here.id);

var A: [1..2, 1..2] real;

coforall loc in Locales {
  on loc {
    var B = A;
  }
}
```

The coforall loop creates
a parallel task per iteration
(in this case, a task per locale)

This results in a distributed parallel computation



Chapel also has other ways of expressing parallelism, not covered today

Low-level:

- **begin:** fires off an asynchronous task
- **cobegin:** creates a fixed number of tasks and waits for them to complete

High-level:

- **foreach:** a way to get vector/SIMD parallelism without using tasks/threads
- **forall:** a parallel loop that divides iterations to tasks (where typically #iters >> #tasks)
  - including zippered loops to iterate over multiple things simultaneously
- **whole-array operations / promotion of scalar operations**
  - equivalent to zippering

The Portability of Chapel's Design over Time

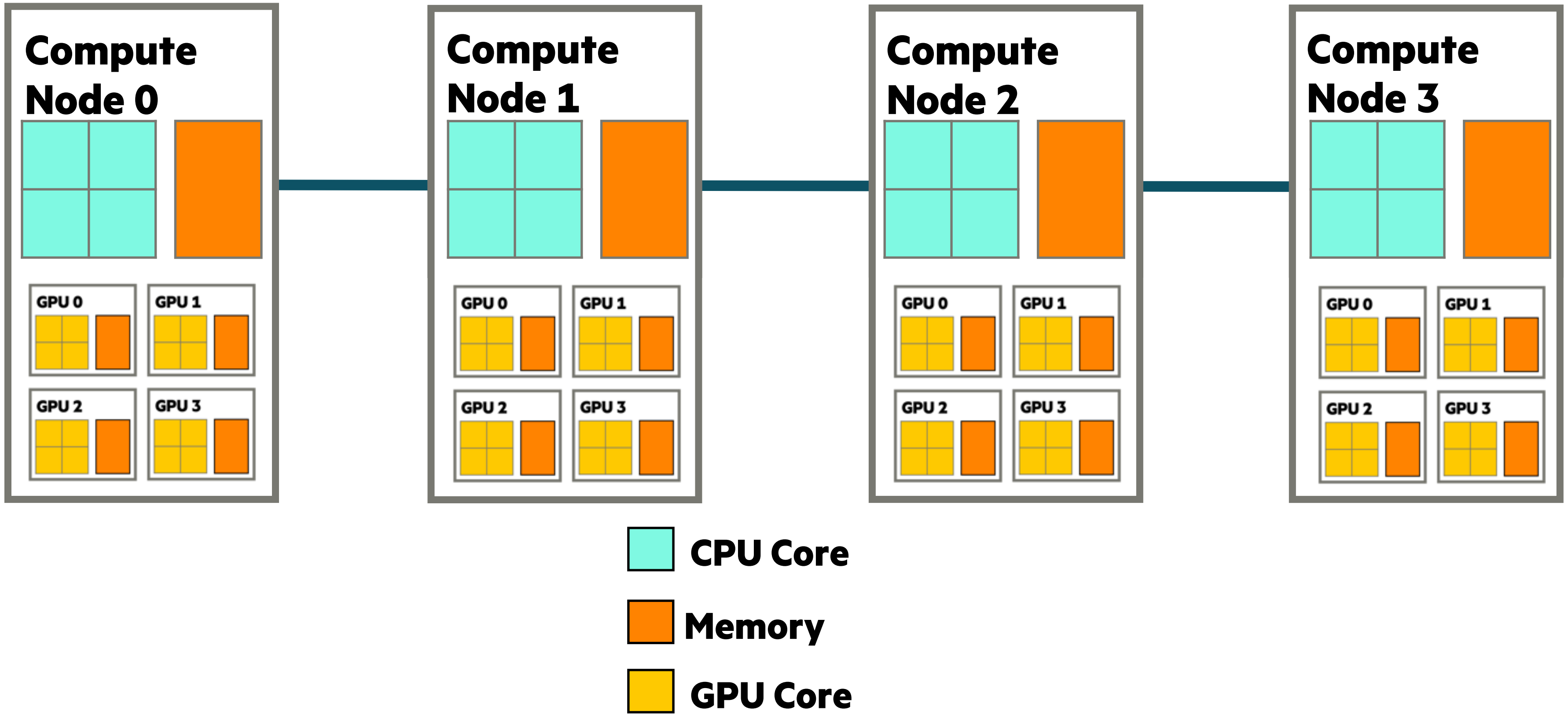
- Chapel's focus on parallelism and locality has made the language design robust to hardware changes

- Consider the timeline:
  - In 2004, multicore CPUs were not yet commonplace or commoditized
  - As a result, Chapel's initial design focused exclusively on:
    - single-core CPU compute nodes
    - the Cray X1
    - the Cray XMT (Tera MTA)

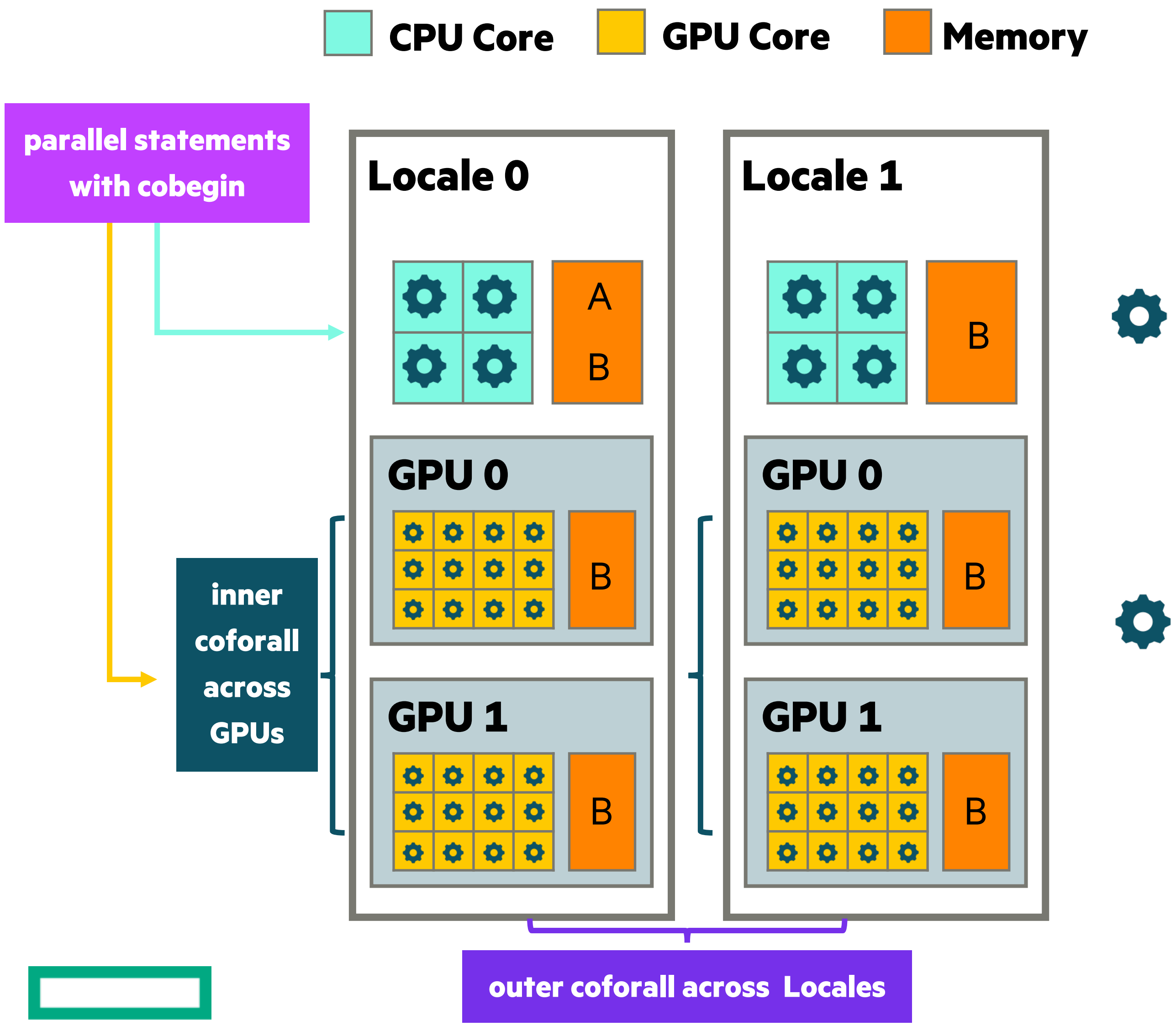
- Chapel's HIPS 2004 features have largely remained unchanged, despite the introduction of:
  - multicore processors
  - multi-socket compute nodes
  - NUMA memory architectures
  - GPUs

Representing GPUs in Chapel

- Modern HPC systems have GPUs
  - And those GPUs have their own cores and memory
  - In Chapel, we represent them as *sub-locales,* using the same locality + parallelism features to program them



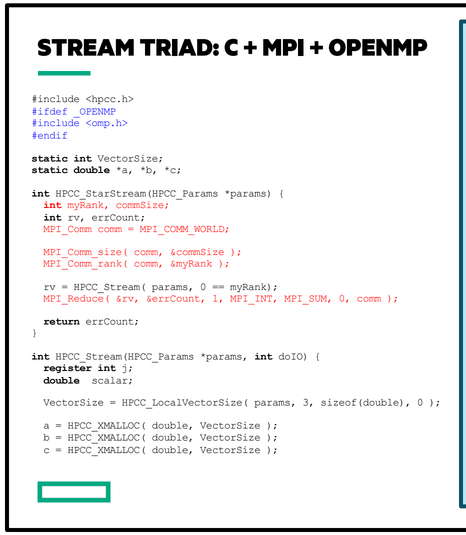
Parallelism and Locality In The Context Of GPUs



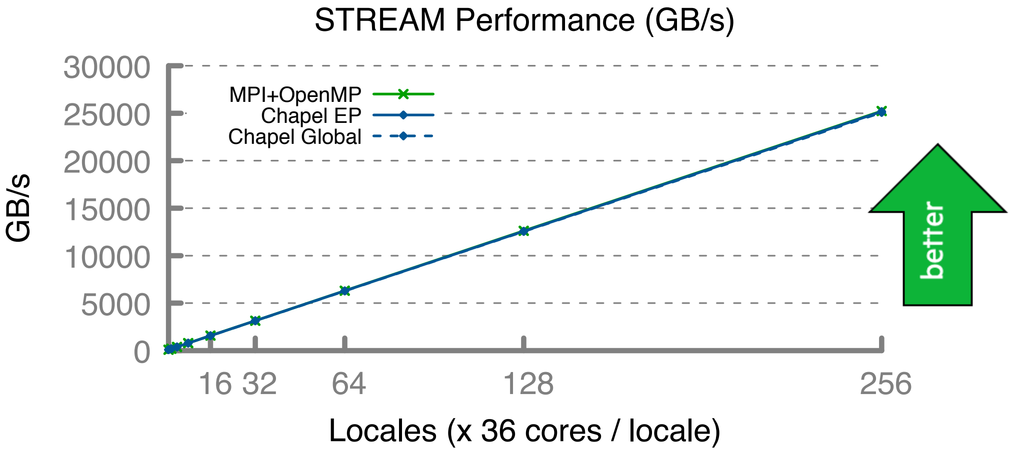

```
var A: [1..n, 1..n] real;
coforall l in Locales do on l {
 cobegin {
  coforall g in here.gpus do on g {
    var B: [1..n, 1..n] real;
    B = 2;
    A = B;
  }
  {
    var B: [1..n, 1..n] real;
    B = 2;
    A = B;
  }
 }
}
writeln(A);
```

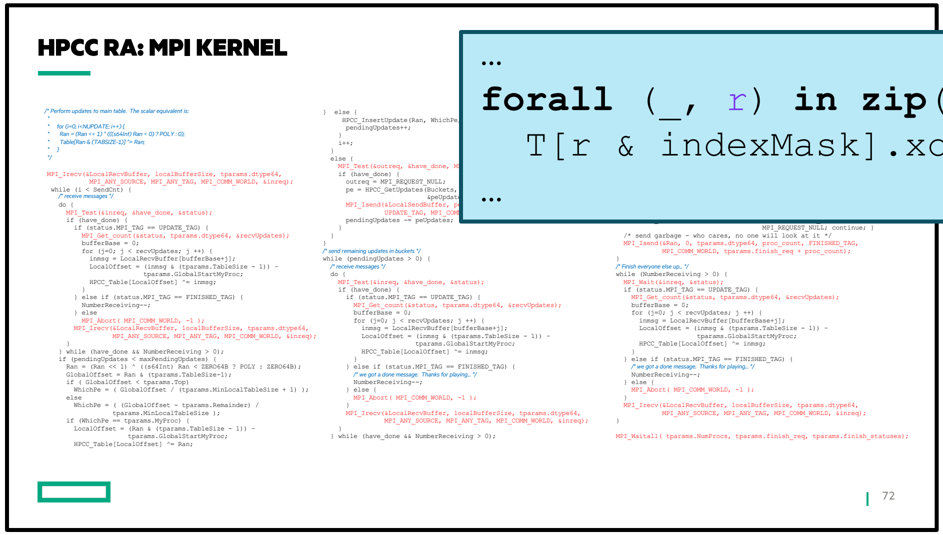
Chapel Benchmarks and Applications

HPCC Stream Triad and RA in C + MPI + OpenMP vs. Chapel

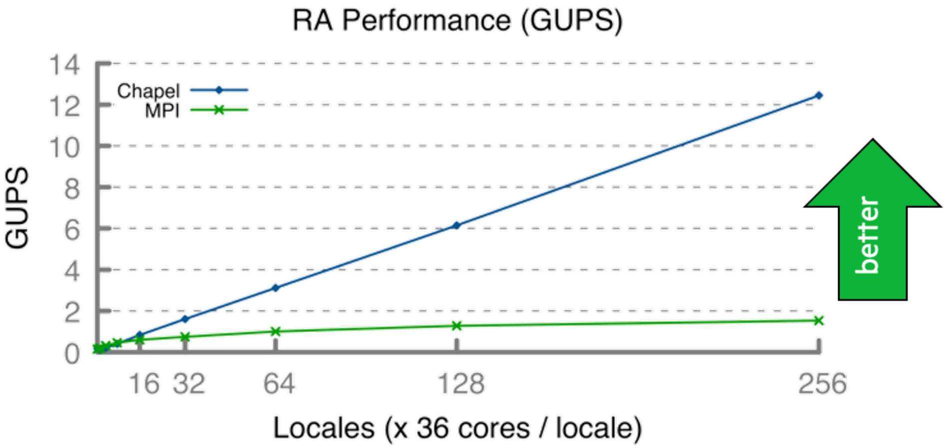



```chapel
use BlockDist;

config const n = 1_000_000,
             alpha = 0.01;
const Dom = blockDist.createDomain({1..n});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```






```chapel
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



Bale IG in Chapel vs. SHMEM on HPE Cray EX (Slingshot-11)

Chapel (Simple / Auto-Aggregated version)

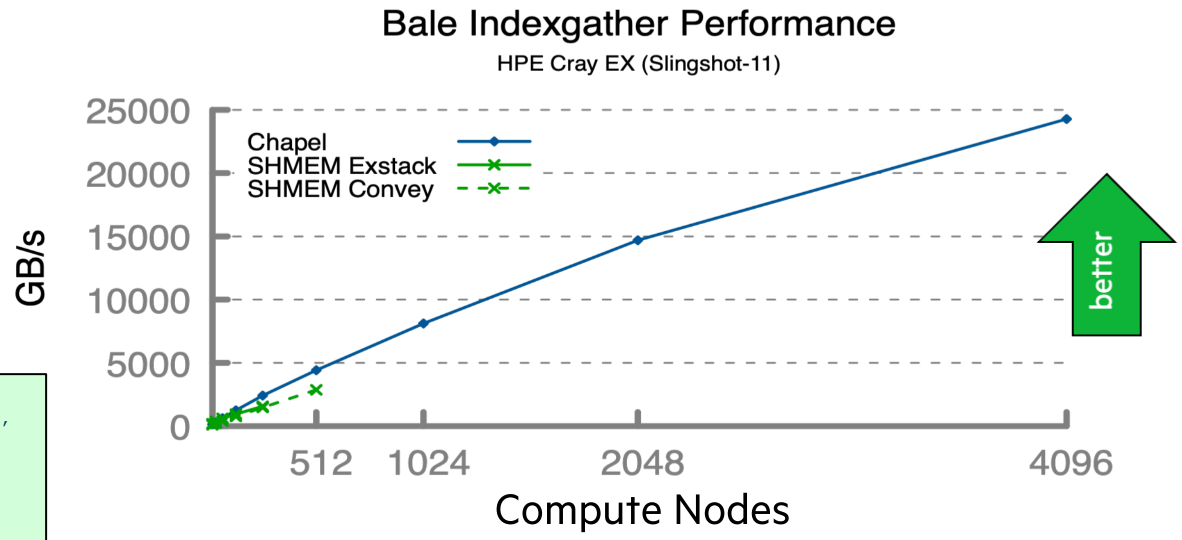
```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

SHMEM (Exstack version)

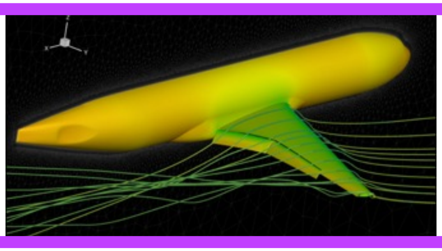
```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_indx = pckindx[i] >> 16;
    pe  = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_indx, pe))
      break;
    i++;
  }

  exstack_exchange(ex);

  while(exstack_pop(ex, &idx , &fromth)) {
    idx  = ltable[idx];
    exstack_push(ex, &idx, fromth);
  }
  lgp_barrier();
  exstack_exchange(ex);

  for(j=i0; j<i; j++) {
    fromth = pckindx[j] & 0xffff;
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

SHMEM (Conveyors version)

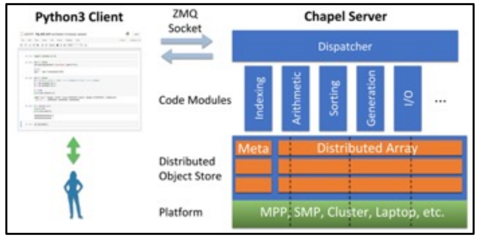
```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

  for (; i < l_num_req; i++) {
    pkg.idx = i;
    pkg.val = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if (! convey_push(requests, &pkg, pe))
      break;
  }

  while (convey_pull(requests, ptr, &from) == convey_OK) {
    pkg.idx = ptr->idx;
    pkg.val = ltable[ptr->val];
    if (! convey_push(replies, &pkg, from)) {
      convey_unpull(requests);
      break;
    }
  }

  while (convey_pull(replies, ptr, NULL) == convey_OK)
    tgt[ptr->idx] = ptr->val;
}
```



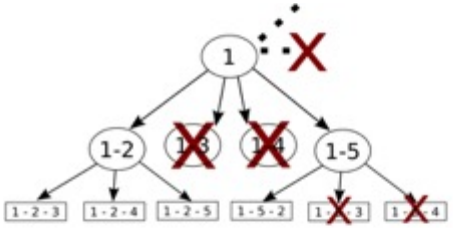
Applications of Chapel



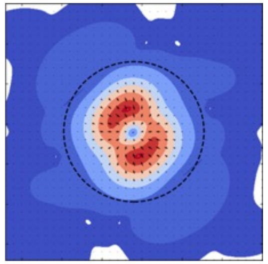
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



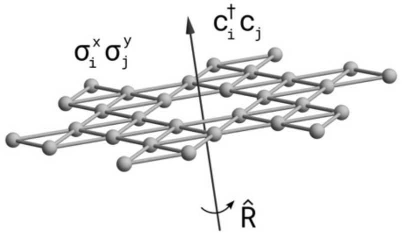
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



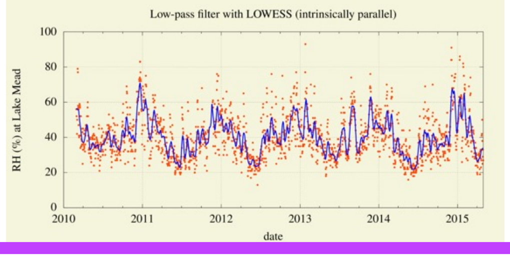
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.

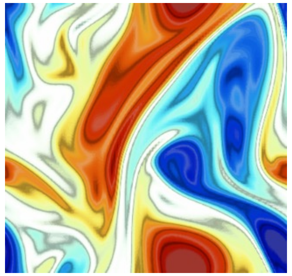


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

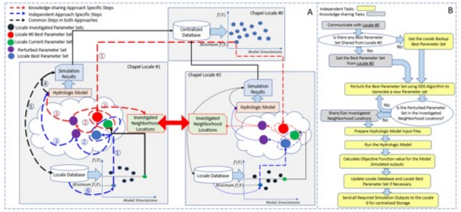


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

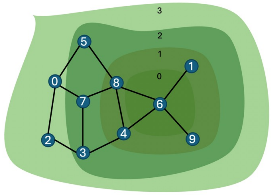
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.

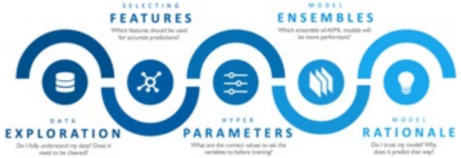


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology

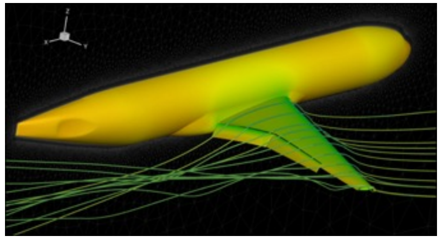
Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

[images provided by their respective teams and used with permission]

Productivity Across Diverse Application Scales (code and system size)



Computation: Aircraft simulation / CFD
Code size: 100,000+ lines
Systems: Desktops, HPC systems

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel
Posted on September 17, 2024.
Tags: Computational Fluid Dynamics | User Experiences | Interviews
By: Engin Kayraklioglu, Brad Chamberlain

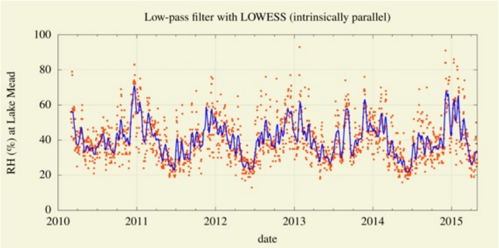
"Chapel worked as intended: the code maintenance is very much reduced, and its readability is astonishing. This enables undergraduate students to contribute, something almost impossible to think of when using very complex software."

Computation: Coral reef image analysis
Code size: ~300 lines
Systems: Desktops, HPC systems w/ GPUs

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel
Posted on October 1, 2024.
Tags: Earth Sciences | Image Analysis | GPU Programming | User Experiences | Interviews
By: Brad Chamberlain, Engin Kayraklioglu

In this second installment of our Seven Questions for Chapel Users series, we're looking at a recent success story in which Scott Bachman used Chapel to unlock new scales of biodiversity analysis in coral reefs to study ocean health using satellite image processing. This is work that

"With the coral reef program, I was able to speed it up by a factor of 10,000. Some of that was algorithmic, but Chapel had the features that allowed me to do it."



Computation: Atmospheric data analysis
Code size: 5000+ lines
Systems: Desktops, sometimes w/ GPUs

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel
Posted on October 15, 2024.
Tags: User Experiences | Interviews | Data Analysis | Computational Fluid Dynamics
By: Engin Kayraklioglu, Brad Chamberlain

In this edition of our Seven Questions for Chapel Users series, we turn to Dr. Nelson Luís Dias from Brazil who is using Chapel to analyze data generated by the Amazon Tall Tower Observatory (ATTO), a project dedicated to long-term, 24/7 monitoring of greenhouse gas fluctuations. Read on

"Chapel allows me to use the available CPU and GPU power efficiently without low-level programming of data synchronization, managing threads, etc."

[read this interview series at: https://chapel-lang.org/blog/series/7-questions-for-chapel-users/]

Gratifying to Have Reached This Point

Larry Snyder:

- "...if we create a great language, we magnify our effort by helping others solve their cool problems."

I'm gratified that we've now done some of this with Chapel

That said, to remain viable, we need to expand from 10's of applications to 100's or 1000's

Why Isn't Chapel More Successful?

Given that Chapel…

- supports such compact, readable code
- has demonstrated performance and scalability
- has been used in such diverse application areas
- has ported across hardware platforms and changes

…why isn't it more broadly adopted?

5 Barriers to HPC Language Adoption

Barrier 1: Creating a practically useful language is a massive effort

- Simplified Chapel timeline:

  2003: Design started

  2008: First public release, not fully featured

  2018: Began encouraging users to try it

- Creating a general-purpose HPC language is strictly harder than creating a traditional language
  - In addition to HPC-crucial aspects…
    - parallelism and locality
    - portability across processors, networks, systems, workload managers, …
    - performance

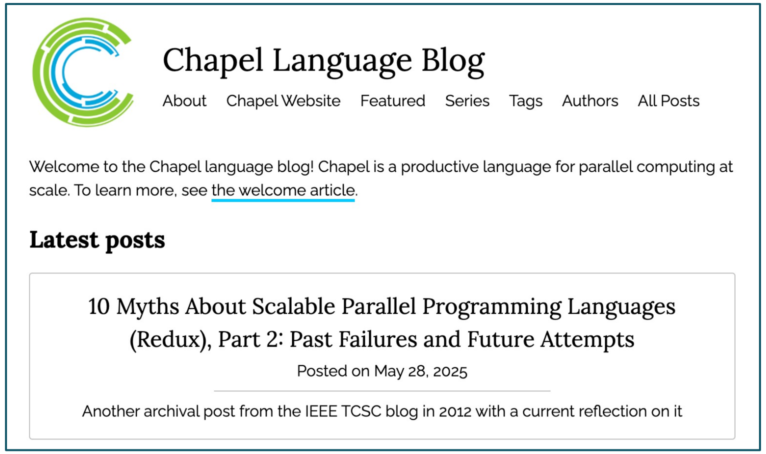
  …users also still want all of the traditional features
    - object-oriented programming
    - error-handling
    - modern memory management
    - productivity features
    - …

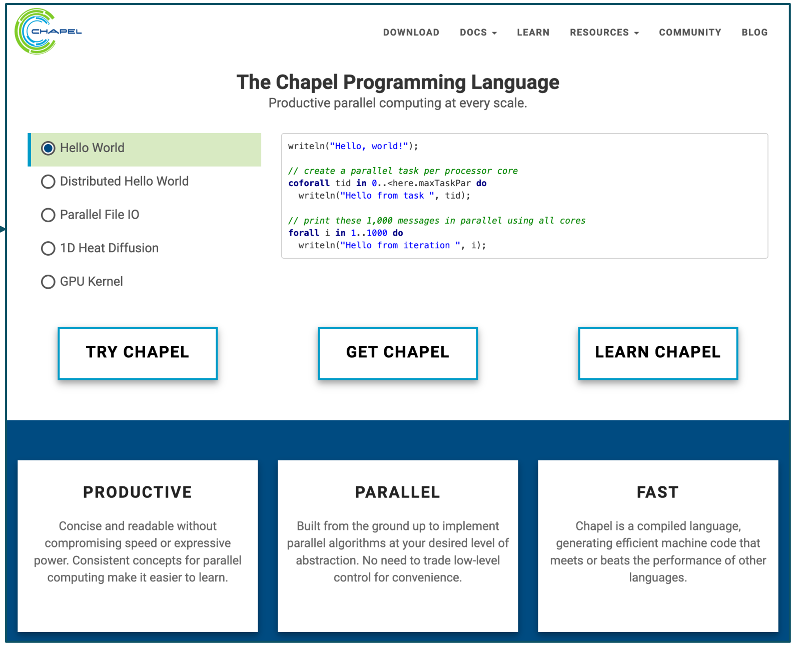
Barrier 2: Human Nature

- Practically speaking, most of us are impacted by limited time, short attention spans, and herd mentality
  - Some evaluated Chapel years ago, when it was not very good/fast/scalable/mature
    - Formed lasting opinions that have never been updated
    - A downside of developing long-term efforts as open-source—growing pains are on display
  - Many will adopt others’ opinions rather than forming their own
  - Many will not adopt a technology until many others have
- Incorrectly assuming “If it hasn’t caught on now, there must be something inherently wrong with it”

Barrier 3: We haven't always marketed ourselves very well

- As a team of R&D engineers, we've often focused on our work and users rather than outreach
  - Chapel's name-recognition isn't as good as it could be as a result
- Have been working on improving this in recent years, by creating:
  - a new website
  - a new blog
  - a better social media presence
  - new community forums
  - …





Barrier 4: HPC Community Behaviors

- Conservative by nature
- “Not invented here” mentalities
- Who makes decisions?
  - Computer scientists or computational scientists?
  - Principal Investigators and money handlers or application programmers?
- Hardware-centric attitudes to the detriment of software, programmers?
- Think of ourselves as a small, niche community
  - E.g., “We’re not big enough to have a language of our own”

Getting HPC Out of its Niche Mentality

Parallel computing has become ubiquitous:

Parallel computing in June 1995:

- supercomputers
- commodity clusters

Parallel computing in June 2025:

- supercomputers
- commodity clusters
- cloud computing
- multicore processors
- GPUs

This gives us an opportunity to leverage the larger community of non-HPC users and use cases

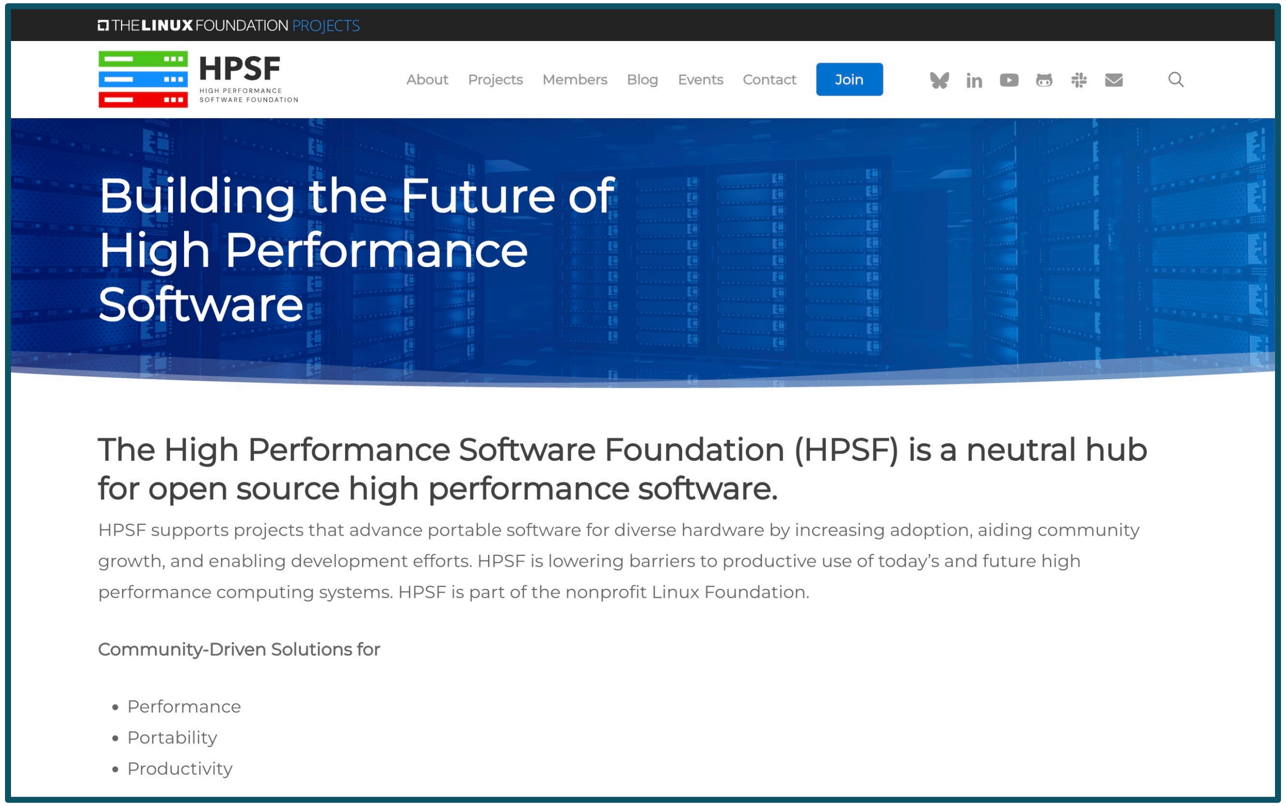
Introduction to HPSF

HPSF = High Performance Software Foundation

- a Linux Foundation project
- a neutral hub for open-source high-perf. software

- **mission:** “to constantly improve the quality and open availability of software for HPC through open collaboration”, focusing on:
  - performance
  - portability
  - productivity

- **goals for member projects:**
  - increasing adoption
  - aiding community growth
  - enabling development efforts



HPSF Timeline and Resources

Timeline:

- **May 2024:** HPSF launched at ISC
- **September 2024:** Began accepting applications for member projects
- **January 2025:** Chapel accepted to HPSF at the "established" project level
- **May 2025:** First-ever HPSFcon

Resources:

- **Website:** https://hpsf.io/
- **Blog:** https://hpsf.io/blog/
- **YouTube channel:** https://www.youtube.com/@HPSF-community
- **GitHub org:** https://github.com/hpsfoundation

HPSF
HIGH PERFORMANCE
SOFTWARE FOUNDATION

Barrier 5: We're increasingly living in a post-programming era

Chapel: "We've developed a great parallel programming language that scales!"

The world:

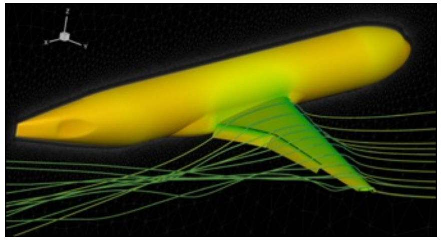
- "Where is the vast set of libraries I'm accustomed to in Python, C++, Julia, …?"
- "Where are all the Stack Overflow articles telling me how to do the things I want to do?"
- "Could an AI write my Chapel code so I don't have to?"

These are very reasonable things to want, but can be difficult to achieve with a small team

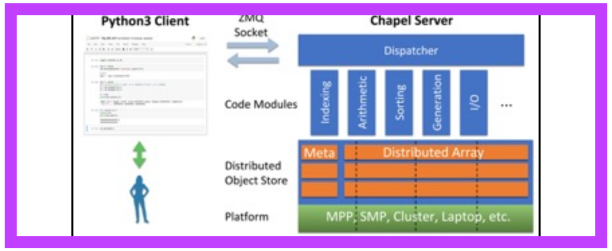
Fortunately, it's also a place where open-source contributors can help out

Arkouda: An HPC Framework for the post-programming world(?)

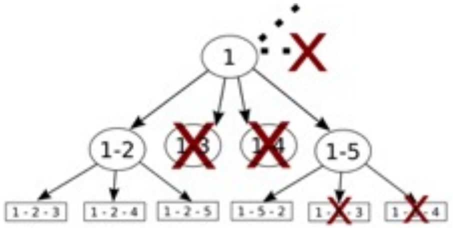
Applications of Chapel



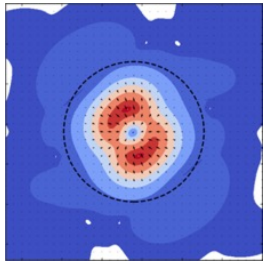
CHAMPS: 3D Unstructured CFD
Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



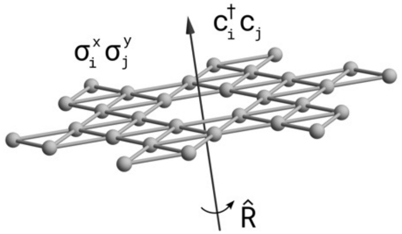
Arkouda: Interactive Data Science at Massive Scale
Mike Merrill, Bill Reus, et al.
U.S. DoD



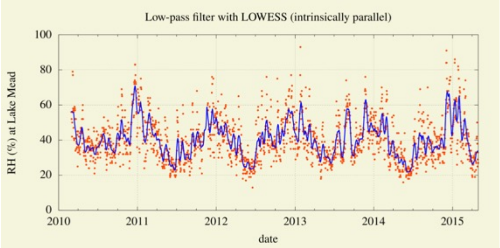
ChOp: Chapel-based Optimization
T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.

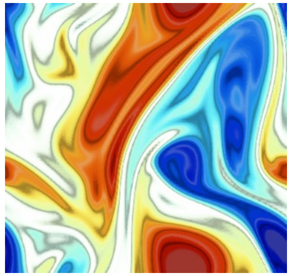


Lattice-Symmetries: a Quantum Many-Body Toolbox
Tom Westerhout
Radboud University

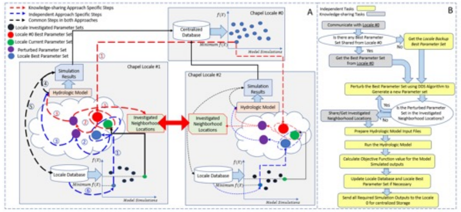


Desk dot chpl: Utilities for Environmental Eng.
Nelson Luis Dias
The Federal University of Paraná, Brazil

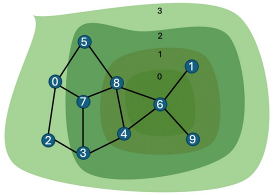
RapidQ: Mapping Coral Biodiversity
Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



ChapQG: Layered Quasigeostrophic CFD
Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.

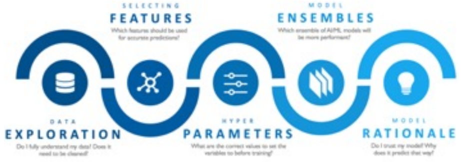


Chapel-based Hydrological Model Calibration
Marjan Asgari et al.
University of Guelph



Arachne Graph Analytics
Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology

Modeling Ocean Carbon Dioxide Removal
Scott Bachman Brandon Neth, et al.
[C]Worthy



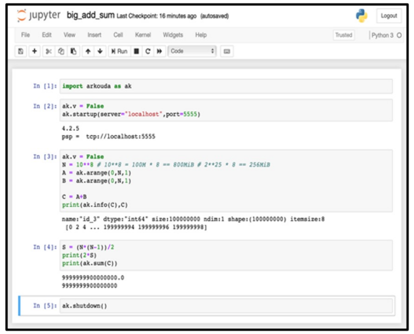
CrayAI HyperParameter Optimization (HPO)
Ben Albrecht et al.
Cray Inc. / HPE

[images provided by their respective teams and used with permission]

What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)

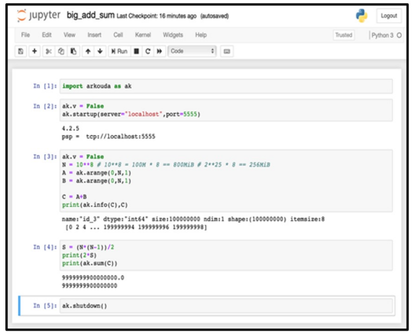


User writes Python code
making familiar NumPy/Pandas calls

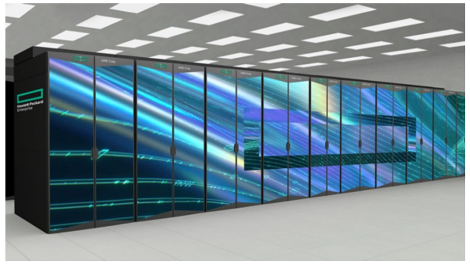
What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

A: “A scalable version of NumPy / Pandas for data scientists”

Performance and Productivity: Arkouda Argsort

HPE Cray EX

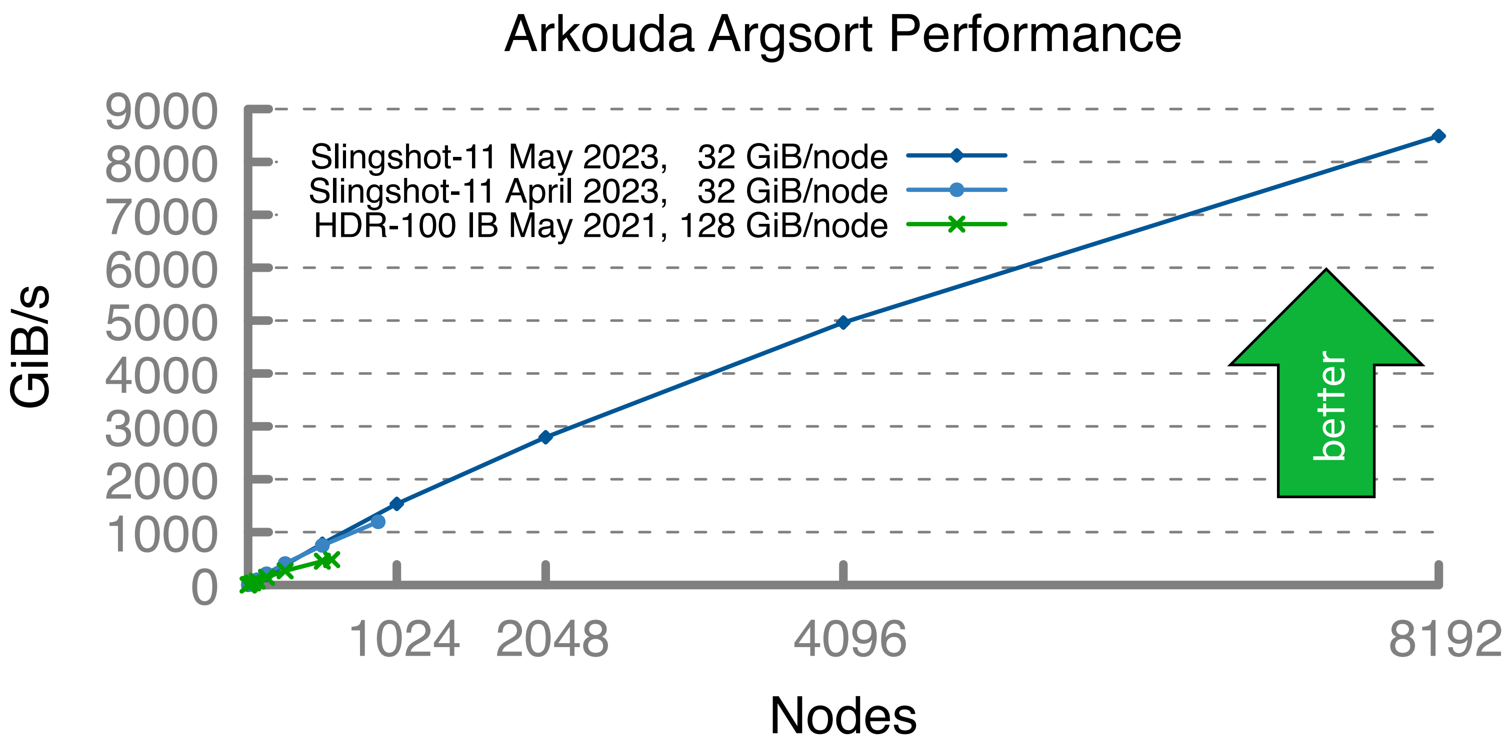
- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

HPE Cray EX

- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

HPE Apollo

- HDR-100 InfiniBand network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

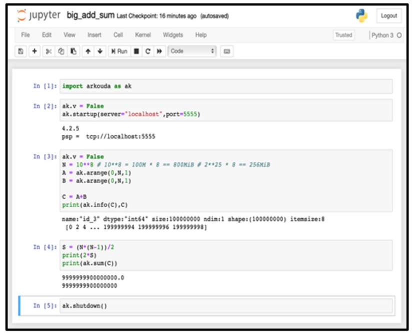


Implemented using ~100 lines of Chapel

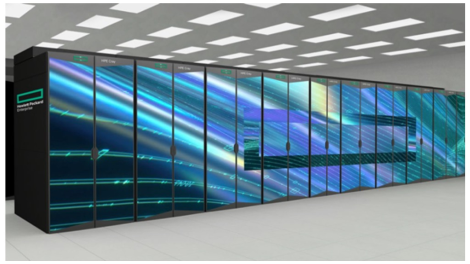
What is Arkouda?

Q: “What is Arkouda?”

Arkouda Client
(written in Python)



Arkouda Server
(written in Chapel)



User writes Python code
making familiar NumPy/Pandas calls

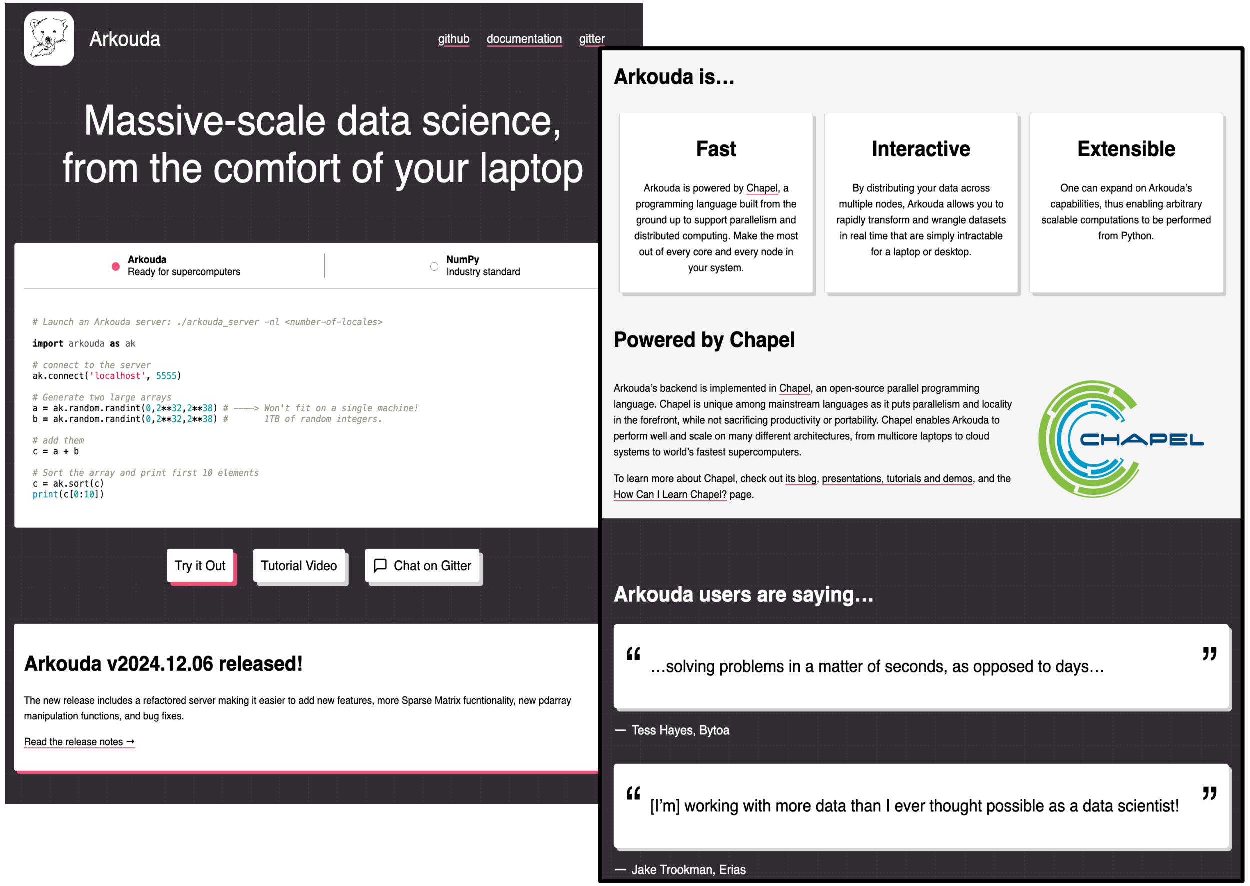
A: “A scalable version of NumPy / Pandas for data scientists”

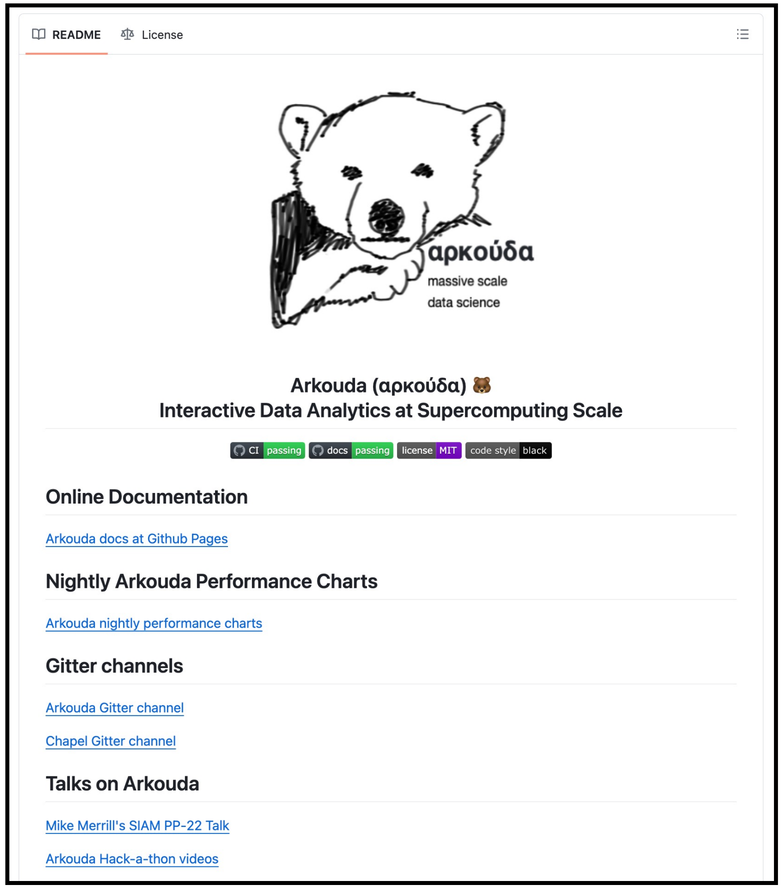
A’: “An extensible framework for arbitrary HPC computations”

A”: “A way to drive HPC systems interactively from Python on a laptop”

Arkouda Resources

Website: https://arkouda-www.github.io/ **GitHub:** https://github.com/Bears-R-Us/arkouda





Arkouda Interview

Blog: Interview with founding co-developer, Bill Reus: https://chapel-lang.org/blog/posts/7qs-reus/

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Bill Reus: Interactive Supercomputing with Chapel for Cybersecurity

Posted on February 12, 2025.

Tags: User Experiences | Interviews | Data Analysis | Arkouda

By: Engin Kayraklioglu, Brad Chamberlain

Table of Contents

We're very excited to kick off the 2025 edition of our Seven Questions for Chapel Users series with the following interview with Bill Reus. Bill is one of the co-creators of Arkouda, which is one of Chapel's flagship applications. To learn more about Arkouda and its support for interactive data analysis at massive scales, read on!

1. Who are you?

My name is Bill Reus, and I live near Annapolis, MD and the beautiful Chesapeake Bay. I am currently a data scientist doing statistical modeling and simulation for the United States government, but I began my career as an experimental chemist. In graduate school, I measured electron transport through thin films of organic molecules using an apparatus that our group invented to collect large volumes of noisy data quickly and with low cost. This approach contrasted with the typical means of studying molecular electronics, which was to spend weeks or months collecting a small number of exquisite measurements in ultra-high vacuum and at ultra-low temperature.

"I was on the verge of resigning myself to learning MPI when I first encountered Chapel. After writing my first Chapel program, I knew I had found something much more appealing."

…

"Chapel's separation of concerns immediately felt like the most natural way to think about large-scale computing. I would highly encourage anyone wanting to get into HPC programming to start with Chapel."

Wrap-up

What can we do to nurture language adoption in HPC?

- Embrace the ubiquity of parallelism and the need for it outside of traditional HPC (cloud, desktop)

- Support open-source efforts and communities like HPSF

- Challenge ourselves to not dismiss technologies we haven't tried firsthand (recently)

- Establish mechanisms for doing trials or comparisons of new HPC software technologies
  - forums for interactions between application programmers and HPC software developers
    - pair programming workshops?
    - co-design sessions?
  - establish frameworks for comparisons
    - HPC equivalent to the Computer Language Benchmarks Game
    - A Top500 equivalent that includes a programming element (HPC Challenge redux?)

- Strive to put HPC software activities on more of an equal footing as hardware

Summary

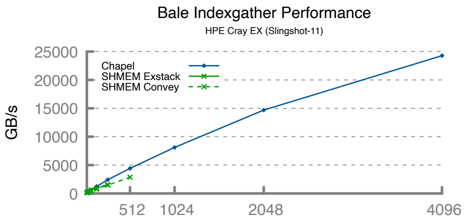
HPC has scaled massively over the past 30 years, but HPC programming hasn’t improved much

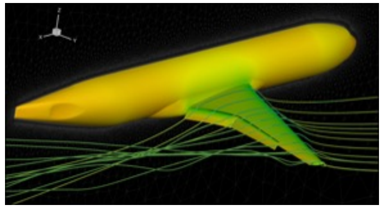
- this can be attributed to the size of the challenge, the nature of our community, and human nature, in part
- non-HPC programming languages show us this need not be the case

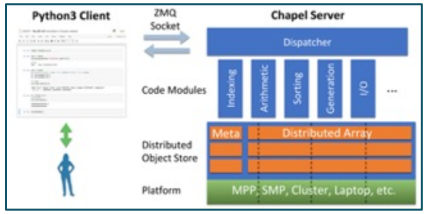
Chapel is unique among programming languages

- built-in features for parallelism and locality
  - make it HPC-ready
  - have kept it timeless despite hardware changes
- ports and scales from laptops to supercomputers
- supports clean, concise code relative to conventional approaches
- supports GPUs in a vendor-neutral manner

```
use BlockDist;

config const n = 10,
             m = 4;

const SrcInds = blockDist.createDomain(0..<n),
      DstInds = blockDist.createDomain(0..<m);

var Src: [SrcInds] int,
    Inds, Dst: [DstInds] int;
…
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

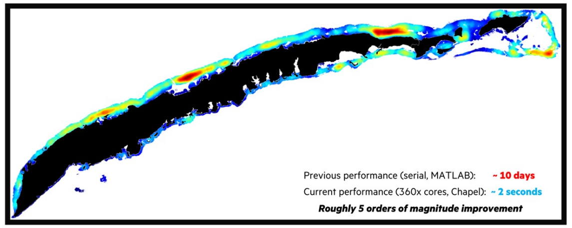






Chapel is being used for productive parallel computing at scale

- users are reaping its benefits in practical, cutting-edge applications
- applicable to domains as diverse as physical simulations and data science
- Arkouda is a particularly unique example of driving HPCs from Python



The Advanced Programming Team at HPE

Ways to Engage with the Chapel Community

“Live” Virtual Events

- ChapelCon (formerly CHIUW), annually
- Project Meetings, weekly
- Deep Dive / Demo Sessions, weekly timeslot

Electronic Communications

- Chapel Blog, ~biweekly
- Community Newsletter, quarterly
- Announcement Emails, around big events

Community / User Forums

- Discord
- Discourse
- Email Contact Alias `chapel+qs@discoursemail.com`
- GitHub Issues
- Gitter
- Reddit
- Stack Overflow

Social Media

- Bluesky
- Facebook
- LinkedIn
- Mastodon
- X / Twitter
- YouTube

Chapel Website

chapel-lang.org

Closing Statement

I consider HPC programmers—current and aspiring—to be as worthy of modern languages as the Python, Swift, Rust, and Julia communities

I believe the number of broadly adopted scalable parallel languages should be ≥1, not the current 0.

Thank you

https://chapel-lang.org
@ChapelLanguage