HPE

30 Years of Scalable Parallel Programming

Brad Chamberlain

PNW PLSE 2026

May 5, 2026

HPE

30 Years of Scalable Parallel Programming: So Many Hardware Advances, So Few Broadly Adopted Languages

Brad Chamberlain

PNW PLSE 2026

May 5, 2026

Terminology for this talk

HPC = High Performance Computing

locality control = the ability to control and/or reason about the placement of data and tasks

- ”allocate this variable in this memory vs. that one”
- “execute this task here vs. there vs. wherever this variable is allocated”

30 Years Ago vs. Now: Top HPC Systems

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

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 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** dragonfly[+], fat-trees (Slingshot, InfiniBand NDR)

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	11,340,000	1,809.00	2,821.10	29,685
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	**JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN EuroHPC/FZJ Germany	4,801,344	1,000.00	1,226.28	15,794
5	**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	

30 Years Ago vs. Now: Top HPC Systems

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

cores: ~563x–141,750x
Rmax: ~3,300,000x–18,300,000x

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** dragonfly[+], fat-trees (Slingshot, InfiniBand NDR)

HPC HW has become far more capable…

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/
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	XP/S-MP 150, Intel DOE/SC/Oak Ridge National Laboratory United States			
4	T3D MC1024-8, Cray/HPE Government United States			
5	VPP500/80, Fujitsu National Lab. for High Energy Physics Japan	80	98.90	128.00

nk	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	El Ca…an - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GH… D Instinct MI300A, Slingshot-11, TOSS, HPE DOE/… …NL Unite…	11,340,000	1,809.00	2,821.10	29,685
	…235a, AMD Optimized 3rd …, AMD Instinct MI250X, …PE …ratory	9,066,176	1,353.00	2,055.72	24,607
	…mpute Blade, …nter GPU	9,264,128	1,012.00	1,980.01	38,698
	…000, GH Superchip …ip, Quad-Rail NVIDIA …terprise Linux, EVIDEN	4,801,344	1,000.00	1,226.28	15,794
5	Eagle… …NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDI… …NVIDIA Infiniband NDR, Microsoft Azure Micro… …ure Unite… …tates	2,073,600	561.20	846.84	

And complex!

- **commodity vector processors**
- **commodity multicore/manycore processors**
- **multi-socket compute nodes**
- **NUMA processor/node architectures**
- **high-radix, low-diameter interconnects**
- **GPU computing**

(Often in ways that hurt programmability)

30 Years Ago vs. Now: Broadly-adopted HPC Programming Notations

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

HPC HW has become far more capable…

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** dragonfly[+], fat-trees (Slingshot, InfiniBand NDR)

Broadly-adopted HPC notations, November 1995:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, vendor-specific pragmas & intrinsics
  - OpenMP on the horizon (1997)
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

Broadly-adopted HPC notations, November 2025:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, SHMEM
- **Intra-node:** Pthreads, OpenMP, Kokkos
- **GPUs:** CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, …
- **Scripting:** Python, bash

30 Years Ago vs. Now: Broadly-adopted HPC Programming Notations

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

HPC HW has become far more capable…

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** dragonfly[+], fat-trees (Slingshot, InfiniBand NDR)

Broadly-adopted HPC notations, November 1995:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, PVM, SHMEM
- **Intra-node:** Pthreads, vendor-specific pragmas & intrinsics
  - OpenMP on the horizon (1997)
- **Scripting:** Perl, sh/csh/tcsh, Tcl/TK

Broadly-adopted HPC notations, November 2025:

- **Languages:** Fortran, C, C++
- **Inter-node:** MPI, SHMEM
- **Intra-node:** Pthreads, OpenMP, Kokkos
- **GPUs:** CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, …
- **Scripting:** Python, bash

HPC SW notations have largely stayed the same, modulo GPU computing and scripting

Notably, HPC has failed to broadly adopt any new programming languages…

Why hasn’t HPC broadly adopted any new programming languages?

Is it because language design is dead?

"Programming language design ceased to be relevant in the 1980s."

—Anonymous reviewer, circa 1995 (paraphrased, from memory)

Seems unlikely...

- Consider all the currently relevant languages that emerged or rose to prominence during those 30 years:
  - **Java** (~1995)
  - **Javascript** (~1995)
  - **Python** (~1991; v2.0 ~2000)
  - **C#** (~2000)
  - **Go** (~2009)
  - **Rust** (~2012)
  - **Julia** (~2012)
  - **Swift** (~2014)

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

That said, they are not particularly HPC-ready—primarily due to lack of locality control

Consider the “hooks” for adopted languages since 1995

Why were they developed? Why did they take hold?

Language	Productivity	Safety	Portability	Performance
Java		✅	✅	
Javascript	✅		✅	
Python	✅			
C#		✅	✅	
Go	✅			✅
Rust		✅		✅
Julia	✅			✅
Swift	✅	✅		✅

Note: lack of a ✅ doesn’t mean a language doesn’t have that quality; just that I don’t consider it a major factor in its design/adoption.

These four themes are extremely valuable in HPC programming as well

Is it because HPC doesn't need new languages?

Libraries, directives, and extensions have obviously gotten us quite far

- Virtually all notable HPC computations from the past 30 years have used them

Yet, using C++ with MPI + OpenMP and/or CUDA leaves a lot to be desired in terms of…

…productivity

…safety

…automated optimizations

We're living in a state that's similar to Fortran's introduction as an alternative to assembly

- moving data around the memory hierarchy manually
- skeptical about higher-level approaches; hesitant to give up control and performance
- undervaluing the benefits of readability, maintainability, safety, compiler checks and optimizations, …

Programming languages offer unique advantages over libraries and extensions

Is it for lack of trying?

Definitely not! Here are some notable attempts from the past 30 years:

- **Mid-to-late 90's Classics:**
  - High Performance Fortran (HPF)
  - NESL
  - Single-Assignment C (SAC)
  - ZPL
- **PGAS founding members:**
  - Coarray Fortran (CAF)
  - Unified Parallel C (UPC)
  - Titanium
- **C-based approaches:**
  - Cilk
  - SAC: Single-Assignment C
- **HPCS-era languages:**
  - Chapel
  - Fortress
  - X10
  - Coarray Fortran 2.0
- **Post-HPCS:**
  - XcalableMP
  - Regent
- **Embedded pseudo-languages**
  - Charm++, Coarray C++, COMPSs, Global Arrays, HPX, Lamellar, Legion, UPC++, …
- **And many more…**

Not all of these attempts have been worthy of broad adoption

Past failures to achieve broad adoption don't mean we should stop trying

What is Chapel?
And how does it fit into this 30-year landscape?

What is Chapel?

Chapel: A scalable parallel programming language

- Open-source & collaborative
  - an HPSF / Linux Foundation project
- Timeline:
  - conceived of in 2002
  - design got traction in 2006
  - first public release in 2008

Chapel and Hardware

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure

Chapel pre-dates all of the HW technology changes mentioned earlier apart from commodity vector processors.

Yet it supports them using the same features for parallelism and locality that we designed 20+ years ago

- **commodity vector processors**
- **commodity multicore/manycore processors**
- **multi-socket compute nodes**
- **NUMA processor/node architectures**
- **high-radix, low-diameter interconnects**
- **GPU computing**

key: express parallelism and locality independently of hardware mechanisms

Chapel, Parallelism, and Locality

Top 5 systems in the Top500, November 1995:

- **Cores:** 80–3,680
- **Rmax:** ~98.9–170 GFlop/s
- **Systems:** Fujitsu, Intel Paragon XP/S, Cray T3D
- **Networks:** crossbar, mesh, 3D torus

Top 5 systems in the Top 500, November 2025:

- **Cores:** 2,073,600–11,340,616
- **Rmax:** 561.2–1809 PFlop/s
- **Systems:** HPE Cray EX, Eviden Bullsequana, Microsoft Azure
- **Networks:** dragonfly[+], fat-trees (Slingshot, InfiniBand NDR)

Broadly-adopted HPC notations, November 1995:

Chapel replaces the need to mix multiple notations:
Uses one language for parallelism at all levels

Broadly-adopted HPC notations, Proposed:

- **Languages:** ~~Fortran, C, C++~~
- **Inter-node:** ~~MPI, SHMEM~~
- **Intra-node:** ~~Pthreads, OpenMP, Kokkos~~
- **GPUs:** ~~CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, …~~
- **Scripting:** Python, bash

key: express parallelism and locality independently of hardware mechanisms

Impact: Succinct Code that Performs and Scales

HPCC Random Access Kernel: MPI

PNW PLSE 2018

Chapel Kernel

```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask] ^= r;
```

MPI Comment

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *     for (i=0; i<NUPDATE; i++) {
 *       Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *       Table[Ran & (TABSIZE-1)] ^= Ran;
 *     }
 */
```

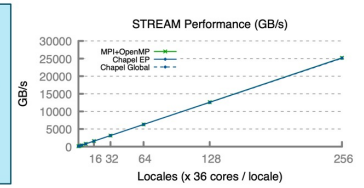
COMPUTE | STORE | ANALYZE

HPC BENCHMARKS: CONVENTIONAL APPROACHES VS. CHAPEL

PNW PLSE 2023

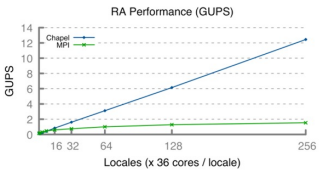
STREAM TRIAD: C + MPI + OPENMP

```
use BlockDist;

config const m = 1000,
             alpha = 3.0;
const Dom = Block.createDomain({1..m});
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



HPCC RA: MPI KERNEL

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



Bale IG in Chapel vs. SHMEM on Cray XC

PNW PLSE 2024

Chapel (Simple / Auto-Aggregated version)

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

Chapel (Explicitly Aggregated version)

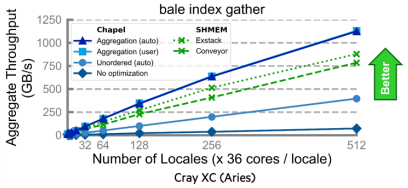
```
forall (d, i) in zip(Dst, Inds) with
 (var agg = new SrcAggregator(int)) do
  agg.copy(d, Src[i]);
```

SHMEM (Exstack version)

```
inds;
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();
}
```

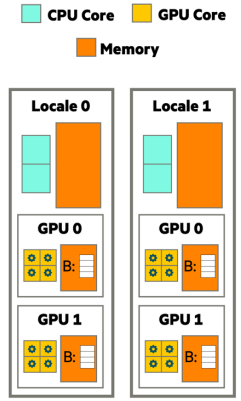
SHMEM (Conveyors version)

```
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;
}
```



Bonus! Hello, Distributed GPUs!

PNW PLSE 2025




```
coforall loc in Locales do on loc {
  coforall gpu in loc.gpus do on gpu {
    var B: [1..10] int;
    B += 1;
  }
}
```

Applications of Chapel

Chapel's proven to be generally applicable & attractive for:

- Computational Fluid Dynamics
- Earth Sciences
- Exploratory Data Analytics
- Astrophysics
- Graph Analysis
- Artificial Intelligence
- …

Chapel Language Blog

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

Posted on October 1, 2024.

Tags: Earth Sciences | Image Analysis | GPUs | User Experiences | Interviews

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics | User Experiences | Interviews

7 Questions for David Bader: Graph Analytics at Scale with Arkouda and Chapel

Posted on November 6, 2024.

Tags: Graph Analytics | Arkouda | User Experiences | Interviews

7 Questions for Akihiro Hayashi: Early Chapel GPU Support through Multiresolution Abstractions

Posted on March 18, 2026.

Tags: GPUs | User Experiences | Interviews | ChapelCon

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

Posted on February 12, 2025.

Tags: Data Analysis | Arkouda | User Experiences | Interviews

7 Questions for Tiago Carneiro and Guillaume Helbecque: Combinatorial Optimization in Chapel

Posted on July 30, 2025.

Tags: Searching / Sorting | GPUs | User Experiences | Interviews

7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel

Posted on September 15, 2025.

Tags: Earth Sciences | User Experiences | Interviews

7 Questions for Oliver Alvarado Rodriguez: Exploiting Chapel's Distributed Arrays for Graph Analysis through Arachne

Posted on January 21, 2026.

Tags: Graph Analytics | Arkouda | Sparse Arrays | User Experiences | Interviews

By: Engin Kayraklioglu, Brad Chamberlain

Part of a series: 7 Questions for Chapel Users

7 Questions for CHAMPS Developers: Empowering Academic R&D to Create Cutting-Edge CFD Apps in Chapel

Posted on March 26, 2026.

Tags: Computational Fluid Dynamics | User Experiences | Interviews

By: Engin Kayraklioglu, Brad Chamberlain

Part of a series: 7 Questions for Chapel Users

7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

Tags: Earth Sciences | Computational Fluid Dynamics | User Experiences | Interviews | Data Analysis

By: Engin Kayraklioglu, Brad Chamberlain

Part of a series: 7 Questions for Chapel Users

So why hasn't HPC broadly adopted any new programming languages? (including Chapel)

Why the stasis in HPC programming languages?

We're a unique community with unique computational needs

We often must focus more on maintaining legacy apps than writing new ones

Our HW::SW investment and focus are heavily biased towards HW

We tend to invent bottom-up programming notations for each new form of HW parallelism, then stop

- vectorization ⇒ vendor-specific pragmas and intrinsics
- distributed memory ⇒ SPMD programming, MPI, SHMEM
- multicore ⇒ Pthreads, OpenMP
- GPUS ⇒ CUDA, HIP, SYCL, OpenMP, Kokkos, OpenACC, OpenCL, …

We seem to doubt that HPC is large / important enough to warrant and sustain a language of its own

We tend not to have frameworks for sustaining HPC software beyond the research stage

And, we still have all the normal social and political challenges of language adoption…

What can we do to nurture language adoption in HPC?

Embrace the ubiquity of parallelism and the need for it outside of traditional HPC

— manycore, cloud, desktop, AI, …

Support open-source efforts and communities like HPSF

— recognize that open-source isn't free

— create funding structures for evolving promising software from research to prototype

Challenge ourselves to **avoid parroting "conventional wisdom"**

— especially about technologies we haven't tried firsthand/ recently

Establish comparison frameworks for new HPC software technologies

— forums for interactions between application programmers and HPC software developers

- pair programming workshops? co-design sessions?

— establish frameworks for comparisons

- An HPC equivalent to the Computer Language Benchmarks Game?
- A Top500 equivalent that includes a programming element? (HPC Challenge redux?)

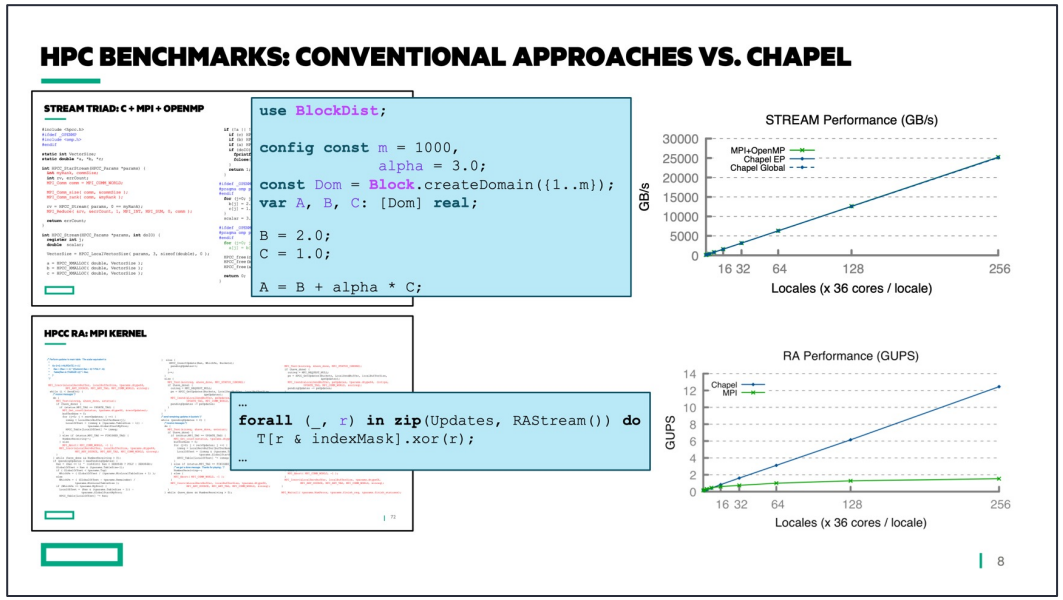
Strive to **reduce the imbalance** between HPC hardware activities and software

AI and Parallel Languages

Q: Now that AIs can program*, do languages like Chapel still have value?

A: I'd say "definitely", at least for the foreseeable future:

- An AI targeting a single, unified parallel language seems likely to be at least as successful as a mash-up of notations
- So long as humans need to check and maintain AI-developed code, the clearer it is the better



(* = your mileage may vary)

Wrapping up...

Closing Statements

I consider current and aspiring scalable parallel programmers to be at least as worthy of modern languages as the Python, Rust, Swift, and Julia communities are

Over the next 30 years, I hope we see the number of broadly adopted languages for scalable parallelism grow to be ≥1, rather than the current 0

For more about…

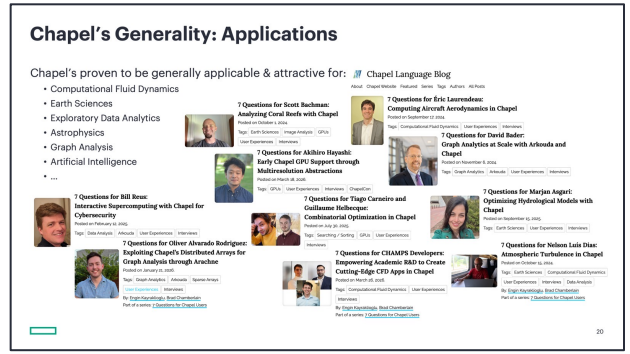
…me grappling with socio-technical aspects of parallel language adoption:

- The blog version of this talk: *Reflections on 30 Years of HPC Programming: So many hardware advances, so little …*
- This thematically related series: *10 Myths About Scalable Parallel Programming Languages (Redux)*
  - Or, start with its summary

…Chapel:

- Attend our talk at the Northwest C++ Users' Group, May 21st @6:30pm, live in Bellevue or virtually
- Browse the Chapel website: Chapel website
- Check out previous PNW PLSE talks
- Reach out to our team and community

…applications of Chapel: Read the *7 Questions for Chapel Users* interview series:



…what's keeping me up at night: See our recent *Chapel Project Seeks New Funding* blog post

Closing Statements

I consider current and aspiring scalable parallel programmers to be at least as worthy of modern languages as the Python, Rust, Swift, and Julia communities are

Over the next 30 years, I hope we see the number of broadly adopted languages for scalable parallelism grow to be ≥1, rather than the current 0

Thank You

@ChapelLanguage