Hewlett Packard Enterprise

CHAPEL: RECENT SUCCESSES, ONGOING CHALLENGES

Brad Chamberlain

DOE Programming Systems Research Forum
February 28, 2022

WHAT IS CHAPEL?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

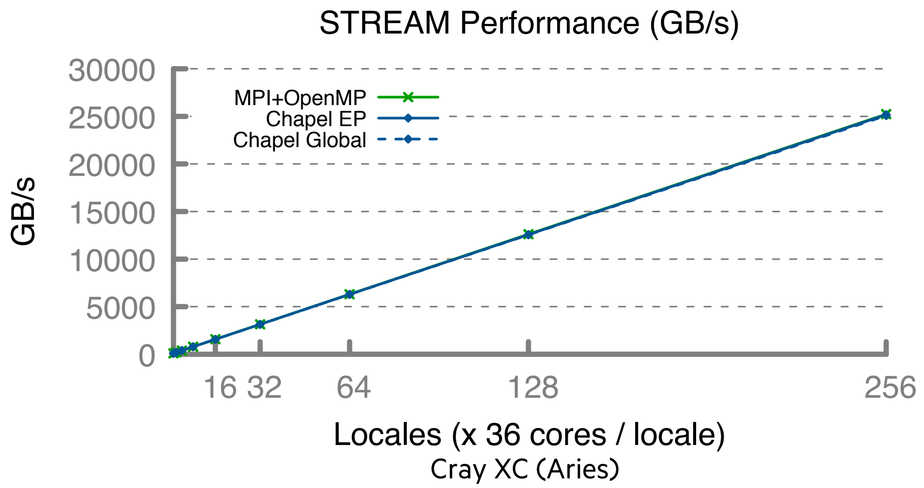
- Support general parallel programming
- Make parallel programming at scale far more productive

FOR HPC BENCHMARKS, CHAPEL TENDS TO BE CONCISE, CLEAR, AND SCALABLE

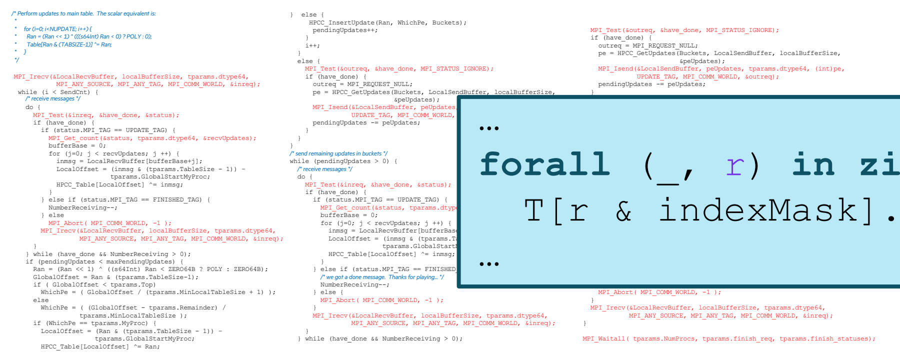
STREAM TRIAD: C + MPI + OPENMP

```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

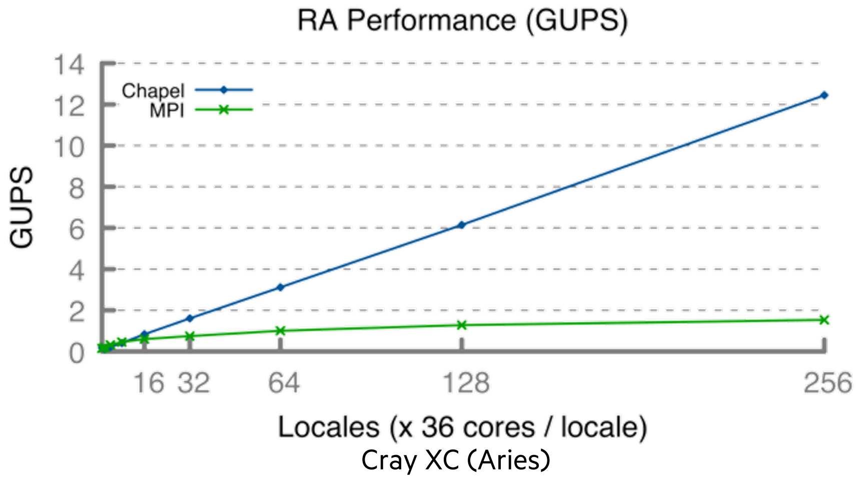
```chapel
use BlockDist;

config const m = 1000,
             alpha = 3.0;

const Dom = {1..m} dmapped …;
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



HPCC RA: MPI KERNEL



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



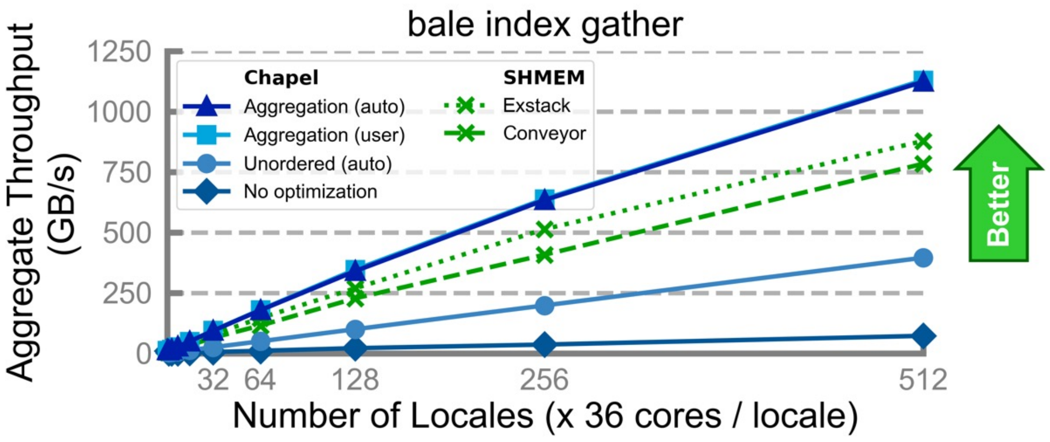
BALE INDEX GATHER: CHAPEL VS. EXSTACK VS. CONVEYORS (JAN 2022)

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();
}
```

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;
}
```



Cray XC (Aries)

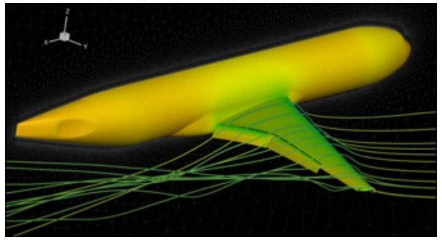
Manually Tuned Chapel version (using aggregator abstraction)

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

Elegant Chapel version (compiler-optimized w/ '--auto-aggregation')

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

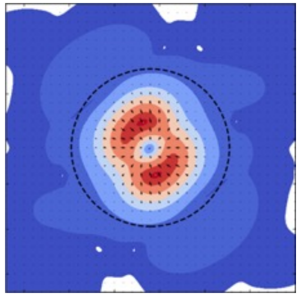
CURRENT FLAGSHIP CHAPEL APPLICATIONS



CHAMPS: 3D Unstructured CFD

Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.

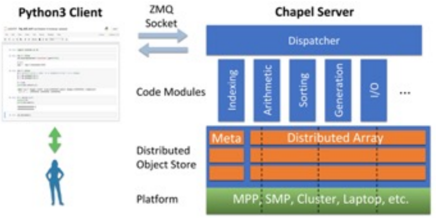
École Polytechnique Montréal



ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, *et al.*

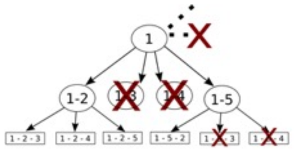
Yale University / University of Auckland



Arkouda: NumPy at Massive Scale

Mike Merrill, Bill Reus, et al.

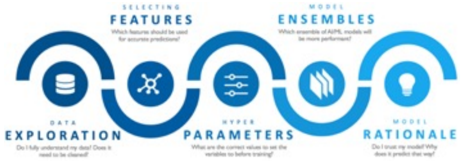
US DoD



ChOp: Chapel-based Optimization

Tiago Carneiro, Nouredine Melab, *et al.*

INRIA Lille, France



CrayAI: Distributed Machine Learning

Hewlett Packard Enterprise

?

Your application here?

THE CHAPEL TEAM

Chapel is a team effort—we're currently at 17.5 full-time employees (+ a director), and we are hiring

Chapel Development Team at HPE

see: https://chapel-lang.org/contributors.html
and https://chapel-lang.org/jobs.html

GOALS OF TODAY'S TALK

1. Provide a Chapel refresher + update on current events
2. Touch on some topics Jeff requested:
   - How to successfully start a language from scratch
   - Identify research challenges

OUTLINE

I. Chapel Context

II. Five Things You Should Know about Chapel

III. Sample Applications of Chapel

IV. The Path to Chapel's "Success"

V. Chapel on GPUs

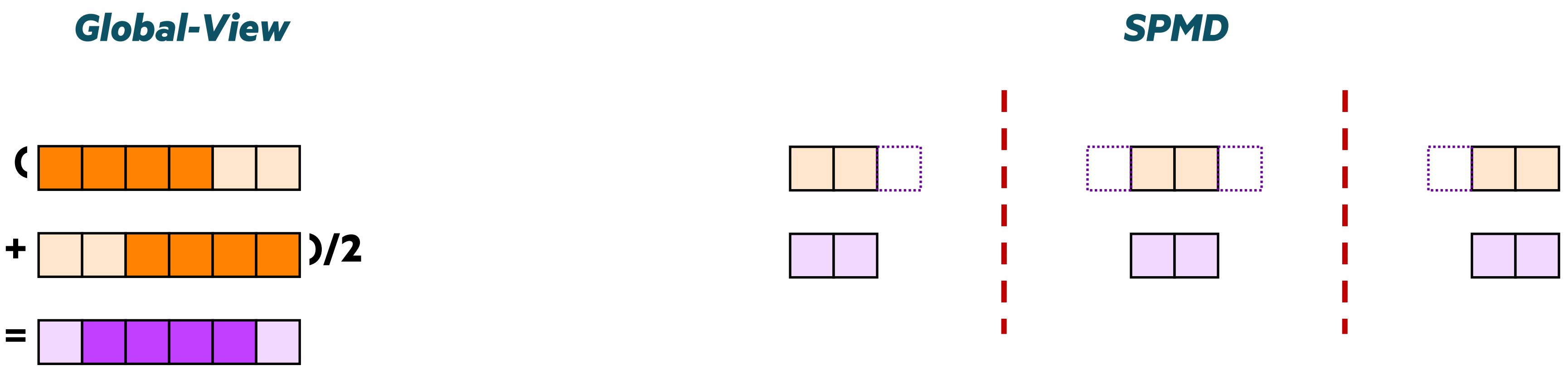
VI. Research Challenges

VII. Wrap-up

FIVE THINGS YOU SHOULD KNOW ABOUT CHAPEL

1) CHAPEL SUPPORTS GLOBAL-VIEW / POST-SPMD PROGRAMMING

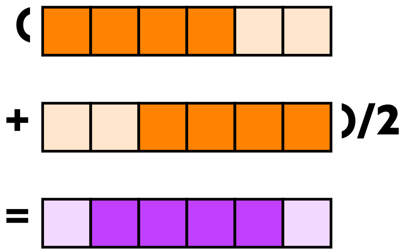
- "Apply a 3-point stencil to a vector"



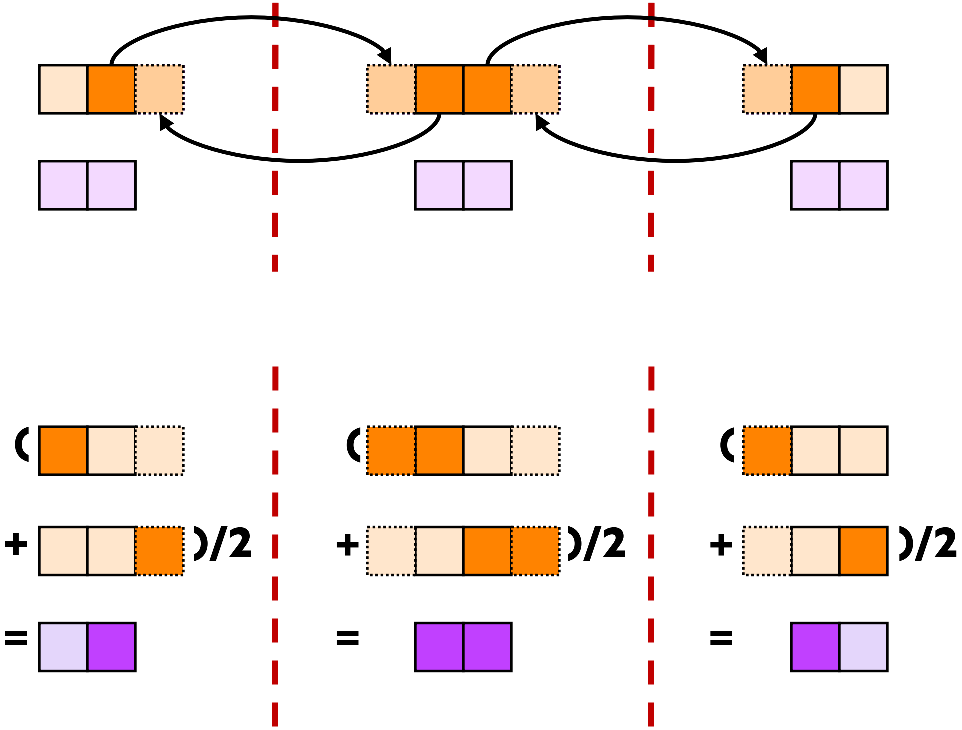
1) CHAPEL SUPPORTS GLOBAL-VIEW / POST-SPMD PROGRAMMING

- "Apply a 3-point stencil to a vector"

Global-View



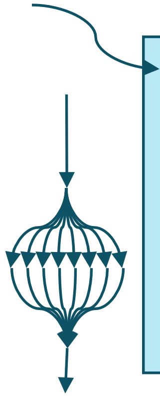
SPMD



1) CHAPEL SUPPORTS GLOBAL-VIEW / POST-SPMD PROGRAMMING

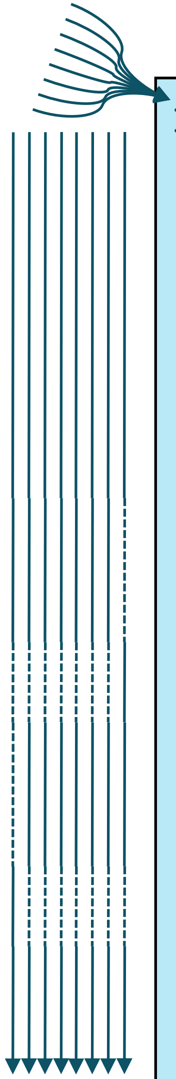
- "Apply a 3-point stencil to a vector"

Global-View Chapel code



```
proc main() {
  var n = 1000;
  var A, B: [1..n] real;

  forall i in 2..n-1 do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

SPMD pseudocode (MPI-esque)



```
proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
      myLo = 1,
      myHi = myN;
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  } else
    myHi = myN-1;
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  } else
    myLo = 2;
  forall i in myLo..myHi do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

2) PARALLELISM AND LOCALITY ARE ORTHOGONAL IN CHAPEL

- This is a parallel, but local program:

```
coforall i in 1..msgs do
  writeln("Hello from task ", i);
```

- This is a distributed, but serial program:

```
writeln("Hello from locale 0!");
on Locales[1] do writeln("Hello from locale 1!");
writeln("Back on locale 0");
```

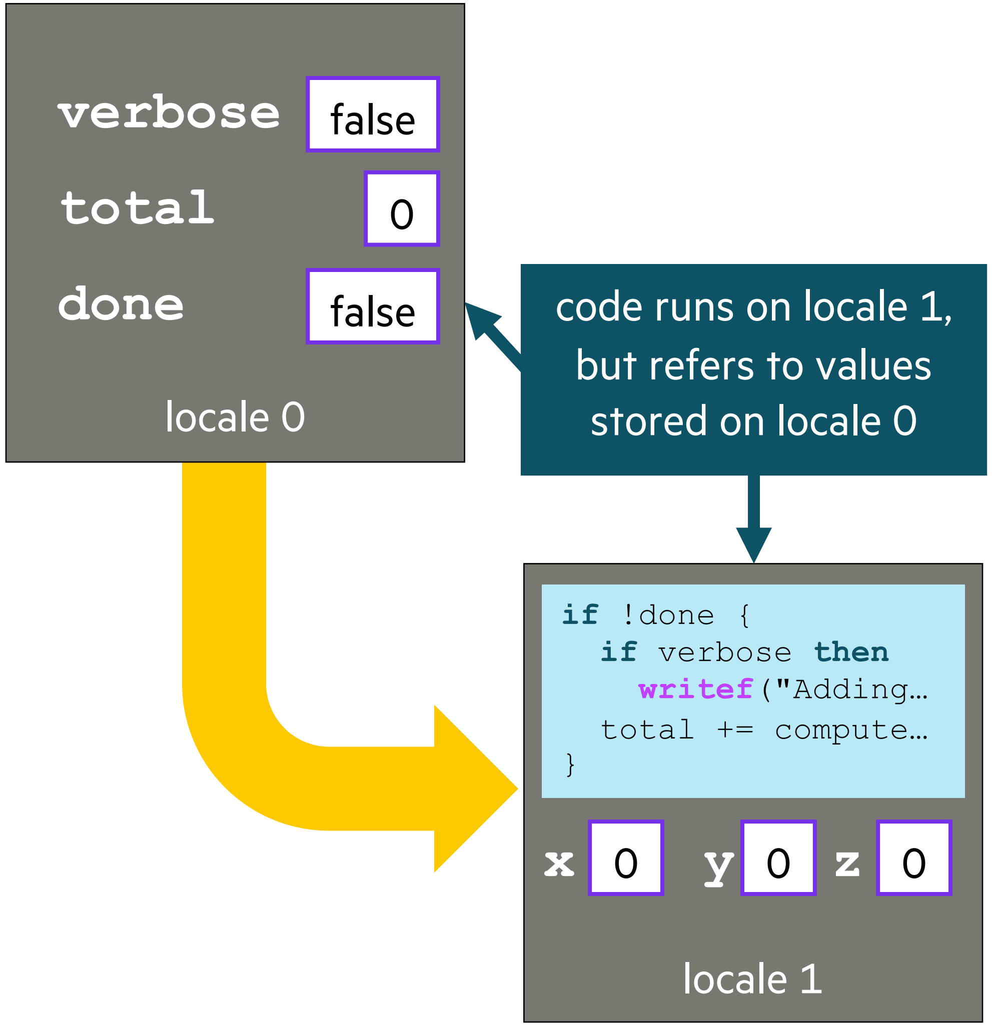
- This is a parallel and distributed program:

```
coforall loc in Locales do
  on loc do
    writeln("Hello from locale ", here.id);
```

3) CHAPEL SUPPORTS A LEXICAL PARTITIONED GLOBAL NAMESPACE

onClause.chpl

```
config const verbose = false;
var total = 0,
    done = false;

…

on Locales[1] {
  var x, y, z: int;
  if !done {
    if verbose then
      writef("Adding locale 1's contribution");
    total += computeMyContribution();
  }
}
```



4) CHAPEL'S COMPILER IS MORE MECHANICAL THAN MAGICAL

```
A = B + alpha * C;
```

whole-array operations are rewritten as zippered forall loops

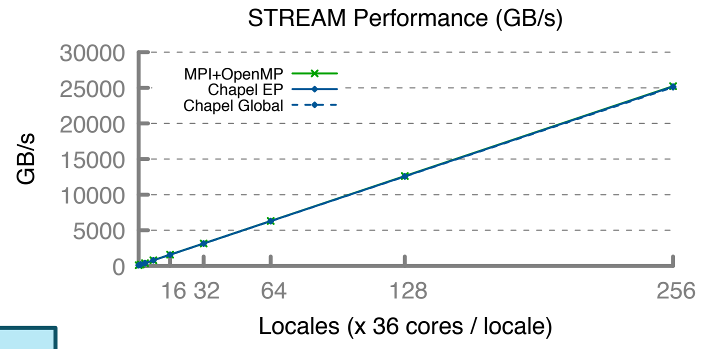
```
forall (a, b, c) in zip(A, B, C) do
  a = b + alpha * c;
```

which are rewritten as invocations of parallel leader-follower iterators

```
forall wk in A.leader() do
  for (a, b, c) in zip(A.follower(wk), B.follower(wk), C.follower(wk) do
    a = b + alpha * c;
```

which are rewritten as explicit tasking and locality constructs (coforall loops and on-clauses)

```
coforall loc in A.targetLocales do       // create a task per locale/node…
  on loc {                               // …resulting in SPMD execution
    const numTasks = here.maxTaskPar;
    coforall tid in 0..<numTasks {       // create a task per core
      const wk = A.myWork(here, tid, numTasks);
      for (a, b, c) in zip(A.follower(wk), B.follower(wk), C.follower(wk) do
        a = b + alpha * c;
    }
  }
```



5) CHAPEL SUPPORTS A MULTIRESOLUTION PHILOSOPHY

- Programmers can mix higher and lower levels of abstraction as desired
  - lowest levels may involve calling out to other languages or embedding C + assembly into Chapel

```
A = B + alpha * C;
```

```
forall (a, b, c) in zip(A, B, C) do
  a = b + alpha * c;
```

```
forall wk in A.leader() do
  for (a, b, c) in zip(A.follower(wk), B.follower(wk), C.follower(wk) do
    a = b + alpha * c;
```

```
coforall loc in A.targetLocales do          // create a task per locale/node…
  on loc {                                  // …resulting in SPMD execution
    const numTasks = here.maxTaskPar;
    coforall tid in 0..<numTasks {          // create a task per core
      const wk = A.myWork(here, tid, numTasks);
      for (a, b, c) in zip(A.follower(wk), B.follower(wk), C.follower(wk) do
        a = b + alpha * c;
    }
  }
```

```
require "blas.h";
extern proc cblas_dgemm(…);
```

```
extern { …my C code… }
```

IMPACTS OF THESE FIVE POINTS

Chapel is general

Chapel is accessible

Chapel is well-architected

IMPACTS OF THESE FIVE POINTS

Chapel is general: With a single, unified language, users can write…

…serial, multicore, or distributed-memory computations

…parallel patterns as simple as SPMD or as complex as is required

…computations for standard CPUs or GPUs

⇒ no need to mix and match multiple, disparate programming models to cover all these cases

Chapel is accessible: Though scalable parallel programming still has inherent challenges, Chapel makes it simpler and far more like traditional programming

- parallelism + locality via language concepts rather than mpirun/aprun, pragmas, etc.

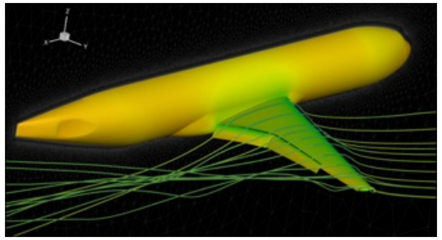
Chapel is well-architected: Built from the system upwards via:

- runtime libraries
- interoperability with C
- low-level, explicit features
- high-level abstractions

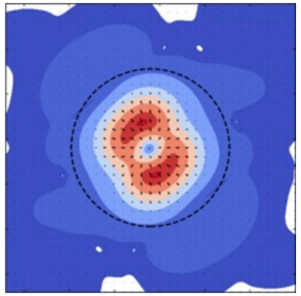
…has resulted in an efficient, complementary, and capable set of features

SAMPLE APPLICATIONS OF CHAPEL

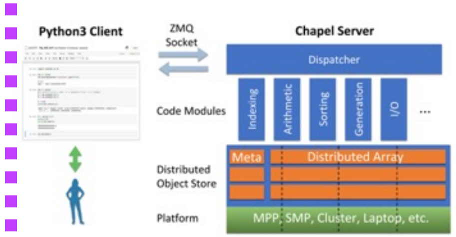
CURRENT FLAGSHIP CHAPEL APPLICATIONS



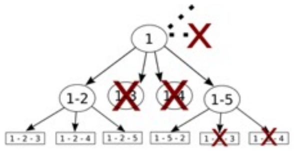
CHAMPS: 3D Unstructured CFD
Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.
École Polytechnique Montréal



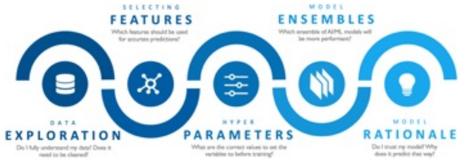
ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan, J. Luna Zagorac, *et al.*
Yale University / University of Auckland



Arkouda: NumPy at Massive Scale
Mike Merrill, Bill Reus, et al.
US DoD



ChOp: Chapel-based Optimization
Tiago Carneiro, Nouredine Melab, *et al.*
INRIA Lille, France



CrayAI: Distributed Machine Learning
Hewlett Packard Enterprise

?

Your application here?

ARKOUDA IN ONE SLIDE

What is it?

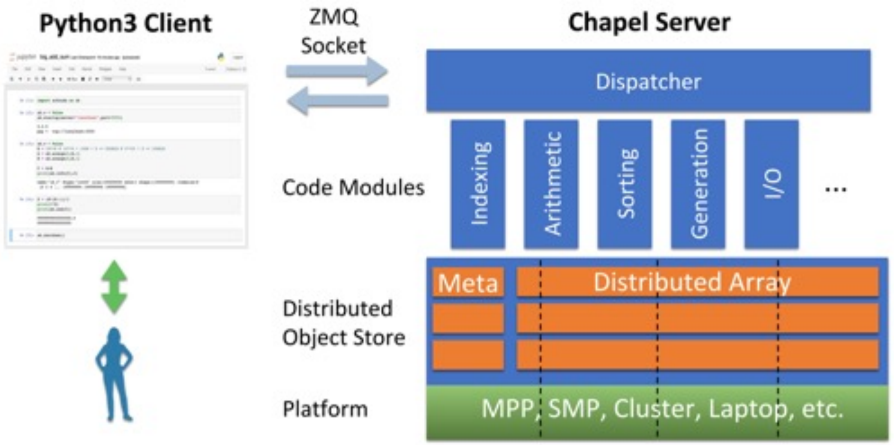
- A Python library supporting a key subset of NumPy and Pandas for Data Science
  - Uses a Python-client/Chapel-server model to get scalability and performance
  - Computes massive-scale results (multi-TB-scale arrays) within the human thought loop (seconds to a few minutes)
- ~19k lines of Chapel, largely written in 2019, continually improved since then

Who wrote it?

- Mike Merrill, Bill Reus, *et al.*, US DoD
- Open-source: https://github.com/Bears-R-Us/arkouda

Why Chapel?

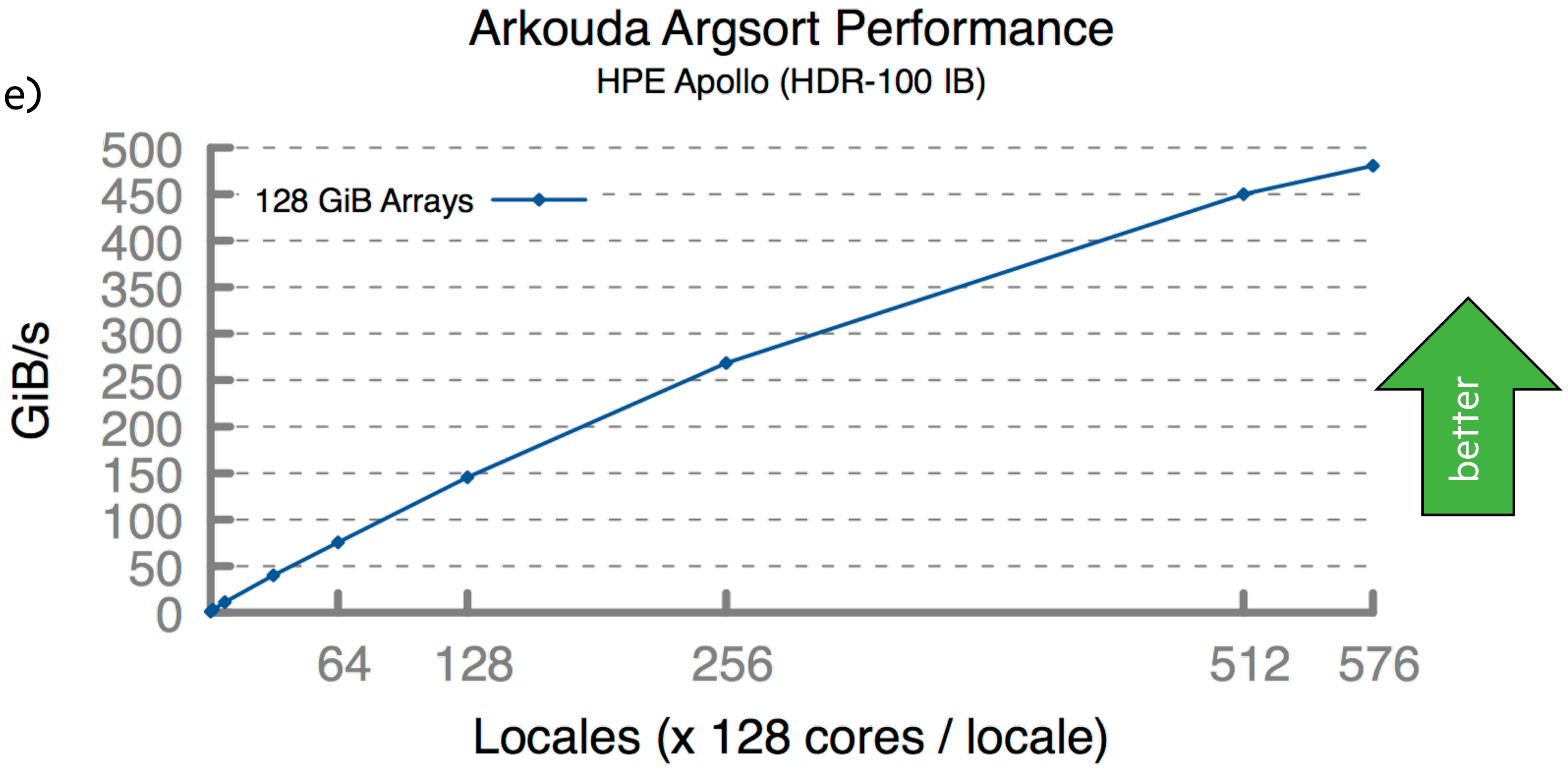
- high-level language with performance and scalability
- close to Pythonic
  - enabled writing Arkouda rapidly
  - doesn't repel Python users who look under the hood
- ports from laptop to supercomputer



αρκούδα
massive scale data science

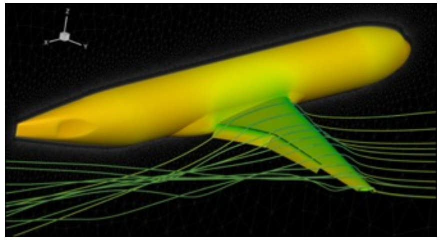
ARKOUDA ARGSORT: HERO RUN

- Recent run performed on a large Apollo system
  - 72 TiB of 8-byte values
  - 480 GiB/s (2.5 minutes elapsed time)
  - used 73,728 cores of AMD Rome
  - ~100 lines of Chapel code



Close to world-record performance (quite likely a record for performance/SLOC)

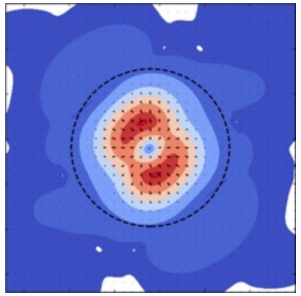
CURRENT FLAGSHIP CHAPEL APPLICATIONS



CHAMPS: 3D Unstructured CFD

Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.

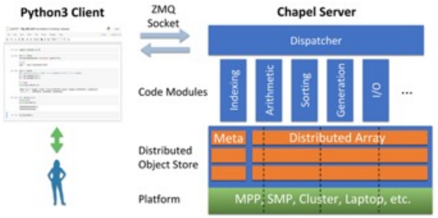
École Polytechnique Montréal



ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, *et al.*

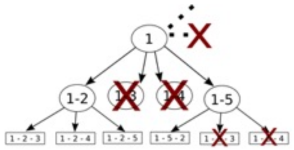
Yale University / University of Auckland



Arkouda: NumPy at Massive Scale

Mike Merrill, Bill Reus, et al.

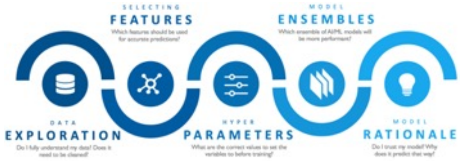
US DoD



ChOp: Chapel-based Optimization

Tiago Carneiro, Nouredine Melab, *et al.*

INRIA Lille, France



CrayAI: Distributed Machine Learning

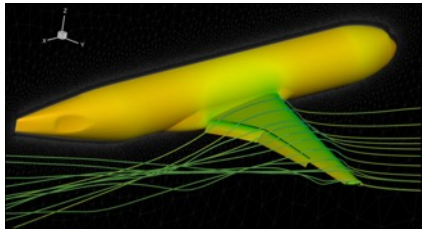
Hewlett Packard Enterprise

?

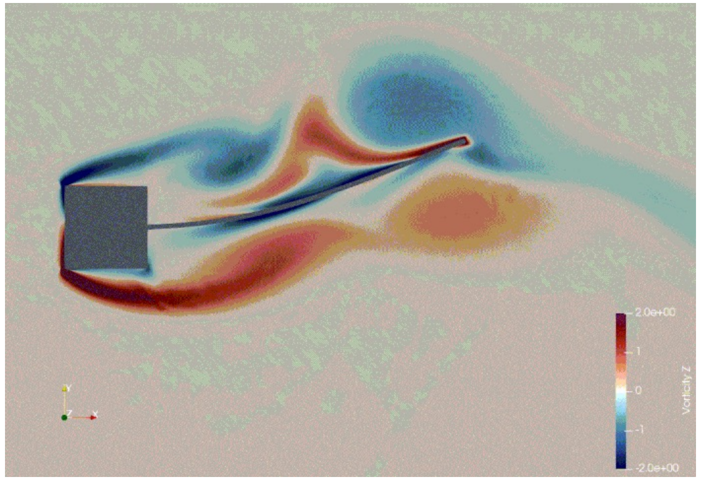
Your application here?

CHAMPS SUMMARY

What is it?



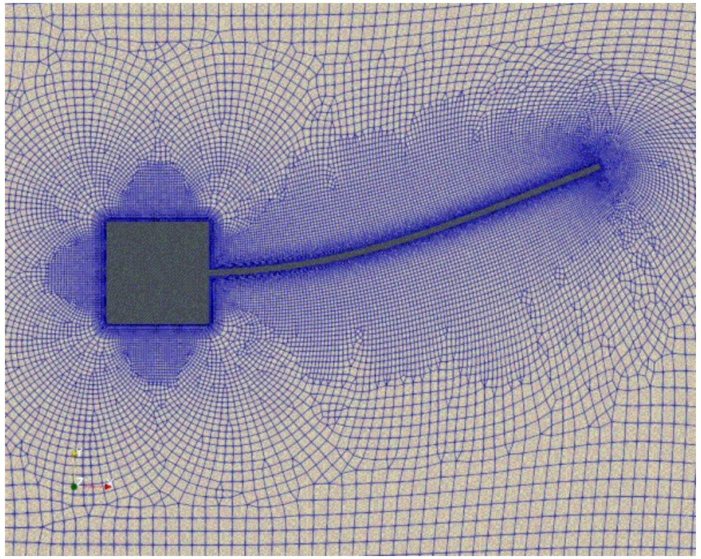
- 3D unstructured CFD framework for airplane simulation
- ~73k lines of Chapel written from scratch in <3 years



Who wrote it?

- Professor Éric Laurendeau’s students + postdocs at Polytechnique Montreal

POLYTECHNIQUE MONTRÉAL



Why Chapel?

- performance and scalability competitive with MPI + C++
- students found it far more productive to use

CHAMPS: EXCERPT FROM ERIC'S CHIUW 2021 KEYNOTE

HPC Lessons From 30 Years of Practice in CFD Towards Aircraft Design and Analysis

"To show you what Chapel did in our lab… [our previous framework] ended up 120k lines. And my students said, 'We can't handle it anymore. It's too complex, we lost track of everything.' And today, they went ***from 120k lines to 48k lines, so 3x less****.*

But the code is not 2D, it's 3D. And it's not structured, it's unstructured, which is way more complex. And it's multi-physics… ***So, I've got industrial-type code in 48k lines.****"*

"[Chapel] promotes the programming efficiency … ***We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months.*** *So, if you want to take a summer internship and you say, 'program a new turbulence model,' well they manage. And before, it was impossible to do."*

"So, for me, this is like the proof of the benefit of Chapel, ***plus the smiles I have on my students everyday in the lab because they love Chapel as well.*** *So that's the key, that's the takeaway."*

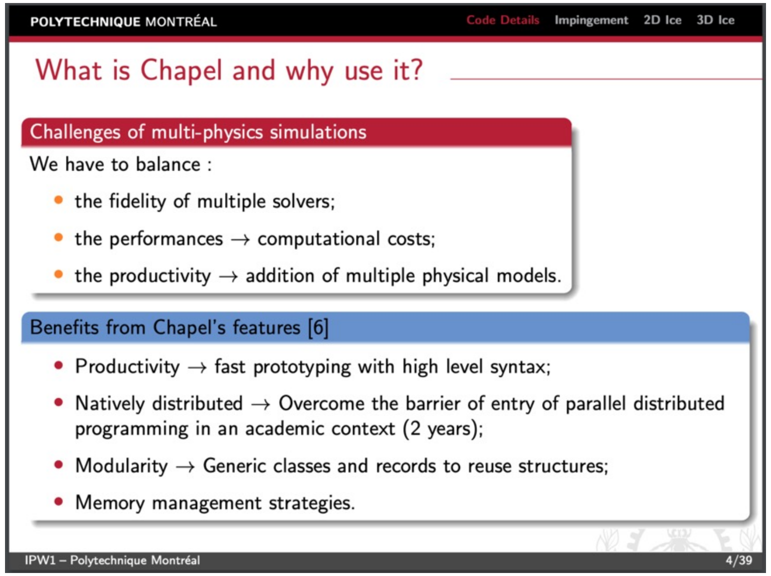
POLYTECHNIQUE MONTRÉAL

- Talk available online: https://youtu.be/wD-a_KyB8aI?t=1904 (hyperlink jumps to the section excerpted here)

CHAMPS 2021 HIGHLIGHTS

- **CHAMPS 2021 Highlights:**
  - Presented at CASI/IASC Aero 21 Conference
  - Participated in 1st AIAA Ice Prediction Workshop
  - Participating in 4th AIAA CFD High-lift Prediction Workshop
  - Student presentation to CFD Society of Canada (CFDSC)

- **Achieving large-scale, high-quality results comparable to other major players in industry, government, academia:**
  - e.g., Boeing, Lockheed Martin, NASA, JAXA, Georgia Tech, …



THE PATH TO CHAPEL'S "SUCCESS"

WHAT THINGS HAVE WE DONE RIGHT?

- Given talks throughout the project's history about our status and plans
  - Have strived to always be brutally honest about the state of our work
- Focused on appealing to end-users and reacting to their feedback
- Didn't fund the development of any of our external flagship apps
- Developed Chapel as open-source and leveraged other open-source projects when appropriate
- Built a team around people who are motivated by the project and technology—not "just a job"
- Not been cowed by the naysayers (and there are a lot in this community)

WHAT THINGS HAVE BITTEN US AT TIMES?

- The impossibility of doing everything everyone might want simultaneously
  - If you can only prioritize one or two of the following at a time, which do you pick, and in what order?
    - documentation
    - fast compilation times
    - GPU support
    - scalar performance
    - publications
    - interoperability
    - scalability
    - portability
    - novel features
    - user support
    - tools
    - interactive programming
- Growing up in the public eye means that people…

  …see you, warts and all

  …may write you off based on early experiences

  …may get numb to your messaging
- Lack of a robust path for turning academic collaborations into production code

WHAT COULD WE HAVE DONE BETTER?

- Demonstrated crossover benefits to mainstream / non-HPC programmers
  - e.g., cloud and multicore users
- Collaborated more with applied scientists rather than computer scientists
- Forged stronger ties with other vendors and DOE users
- Focused on GPUs earlier (or at least, the right ones)
- …? (I'm too close to see everything, and would be curious for others' constructive observations)

12 STEPS FOR CREATING A "SUCCESSFUL" LANGUAGE FROM SCRATCH

	Step	Chapel's approach in a nutshell
✓	1. Identify language's motivation and novelty	Productive language support for parallelism & locality
✓	2. Pitch & prototype features to gain mindshare	HPCS phase II & III milestone reviews, SW productivity meetings
✓	3. Solve core research problems in the design	unify task & data parallelism, multiresolution design, …
✓	4. Leverage third-party code as useful/appropriate	GASNet, Qthreads, hwloc, jemalloc, GMP, LLVM, libfabric, …
✓	5. Improve prototype to illustrate approach's benefits	user-defined distributed arrays, zippered forall loops, LULESH, …
✓	6. Get performance to a competitive state	Stream Triad, HPCC RA, ISx, Bale Indexgather, …
✓	7. Backfill non-researchy language/library features	standard libraries, error-handling, package manager, OOP, …
✓	8. Be attentive and responsive to users	mailing lists, web forums, StackOverflow, Gitter, telecons, …
✓	9. Hope some flagship applications use your language	Arkouda, CHAMPS, ChplUltra, ChOp, CrayAI, …
…	10. Improve the user experience	faster / separate compilation, better tools, integration into editors
…	11. Extend language's applicability	extending Chapel to codegen for GPUs, support AWS, FAM, …
?	12. Reach a point of sufficiently broad adoption to not have to continually justify your existence	TBD

CHAPEL ON GPUS

FAQ: DOES CHAPEL SUPPORT GPUS?

Maybe? Ten years of research about Chapel and GPUs (UIUC, AMD, Georgia Tech, …)

Yes: Apps like CHAMPS and ChOp use GPUs from Chapel via interoperability

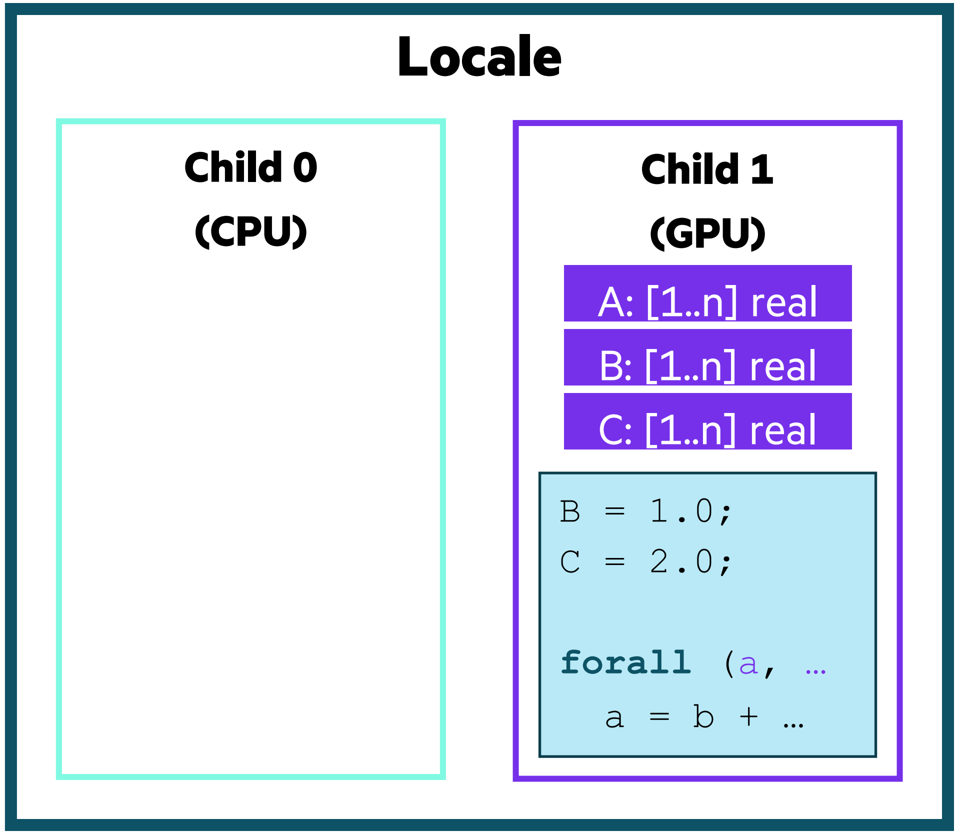
No: Our production compiler has never supported GPUs through code generation, despite potential

Until Chapel 1.25! (fall 2021)

STREAM TRIAD FOR GPUS IN CHAPEL

Chapel 1.25 added support for simple GPU computations via on-clauses and forall loops:

```
on here.getChild(1) {
  var A, B, C: [1..n] real;
  const alpha = 2.0;

  B = 1.0;
  C = 2.0;

  forall (a, b, c) in zip(A, B, C) do
    a = b + alpha * c;
}
```

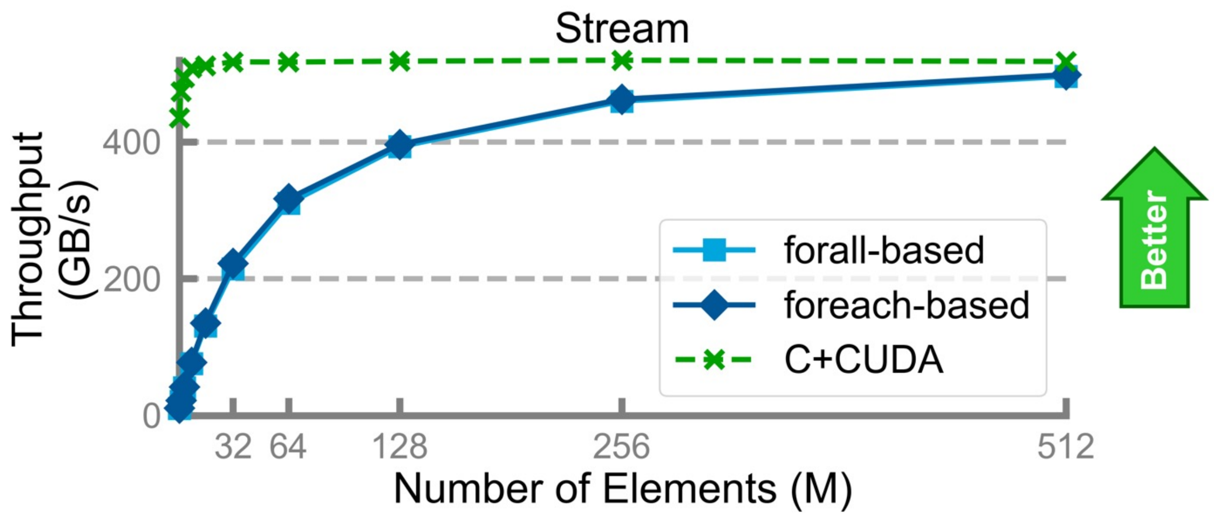


developed by Engin Kayraklioglu, Andy Stone, and David Iten

STREAM TRIAD FOR GPUS IN CHAPEL

Chapel 1.25 added support for simple GPU computations via on-clauses and forall loops:

```
on here.getChild(1) {
  var A, B, C: [1..n] real;
  const alpha = 2.0;

  B = 1.0;
  C = 2.0;

  forall (a, b, c) in zip(A, B, C) do
    a = b + alpha * c;
}
```



developed by Engin Kayraklioglu, Andy Stone, and David Iten

RESEARCH CHALLENGES

1. Language Support for GPUs / Accelerators
2. Tools for New Languages
3. Socio-political Challenges to Adoption

RESEARCH CHALLENGES: LANGUAGE SUPPORT FOR GPUS/ACCELERATORS

Representation in programming languages

- how to represent accelerators and their processors/memories relative to host CPUs/memory?
- how to represent relationship between #sockets, #accelerators, #NICs

Implementation challenges

- extending the partitioned global namespace to include accelerators
- making data movement effective and efficient (local host⟷GPU x across compute nodes)

Portable code generation

- across vendors
- across architectural generations within a single vendor

Extensible, future-proof design, particularly for forthcoming accelerators that may be less GPU-like

- ability to represent future capabilities in the language
- ability to target future accelerators from the compiler
  - ideally, by external developers

RESEARCH CHALLENGES: TOOLS FOR NEW LANGUAGES

If/when one or more languages are adopted, how will tools support them?

- Can existing tools be adapted to work with them?
- Are new tools or capabilities required?
- What differences or commonalities across programming models do tools need to be aware of?
- Are there things that a new language could do to aid tool developers?

RESEARCH CHALLENGES: SOCIO-POLITICAL CHALLENGES TO ADOPTION

How to overcome socio-political challenges to evaluation and adoption?

- How to evaluate/compare/contrast various strengths and weaknesses in a neutral setting?
- How to generate the time / incentive for applied scientists to investigate alternative programming languages?
  - "If you build it, will they necessarily come?" [If you're not paying/directing them to do so?]
- How to get distinct PIs / vendors working together on common efforts?

A few potential ideas:

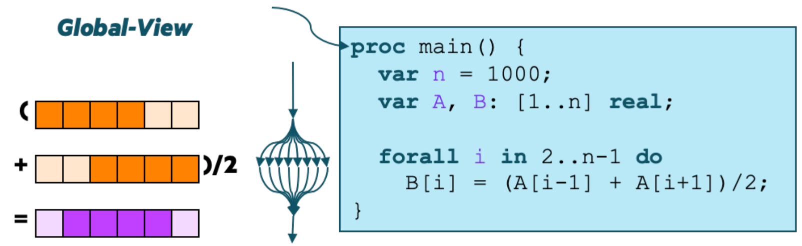
- HPC framework similar to the Computer Language Benchmarks Game?
- hands-on speed-dating workshop between applied scientists and programming model teams?
- programming model institute?

WRAP-UP

SUMMARY

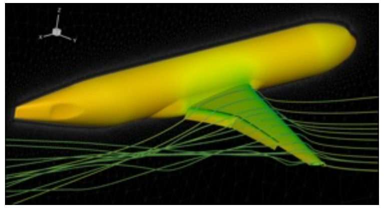
Chapel is unique among HPC programming models

- its post-SPMD nature, lexical PGNS, and multiresolution philosophy make HPC programming far more accessible, without throwing away control or performance

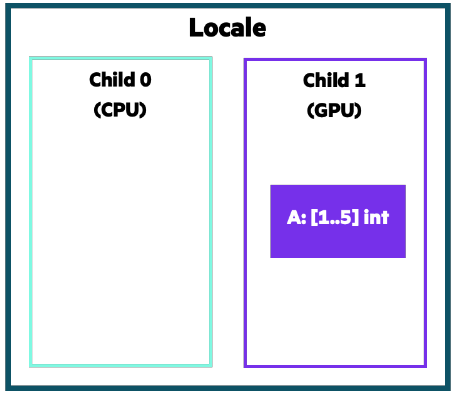


Chapel is being used for productive parallel programming at scale

- users are reaping its benefits in 19k–73k-line applications



Though significant challenges remain in targeting GPUs with Chapel, support is improving by leaps and bounds



We'd enjoy engaging with DOE employees who'd like to know more

CHAPEL RESOURCES

Chapel homepage: https://chapel-lang.org

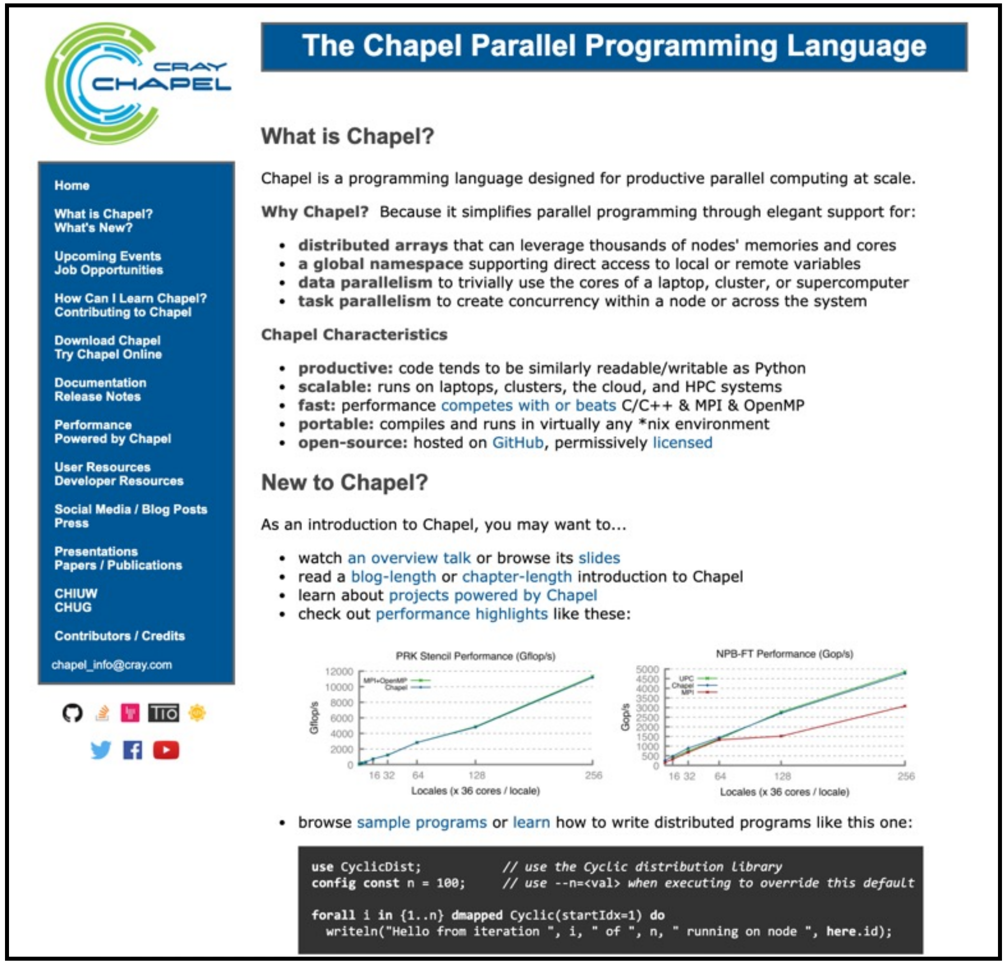
- (points to all other resources)

Social Media:

- Twitter: @ChapelLanguage
- Facebook: @ChapelLanguage
- YouTube: http://www.youtube.com/c/ChapelParallelProgrammingLanguage

Community Discussion / Support:

- Discourse: https://chapel.discourse.group/
- Gitter: https://gitter.im/chapel-lang/chapel
- Stack Overflow: https://stackoverflow.com/questions/tagged/chapel
- GitHub Issues: https://github.com/chapel-lang/chapel/issues



SUGGESTED READING / VIEWING

Chapel Overviews / History (in chronological order):

- *Chapel* chapter from *Programming Models for Parallel Computing*, MIT Press, edited by Pavan Balaji, November 2015
- *Chapel Comes of Age: Making Scalable Programming Productive*, Chamberlain et al., *CUG 2018*, May 2018
- Proceedings of the *8th Annual Chapel Implementers and Users Workshop* (CHIUW 2021), June 2021
- Chapel Release Notes — current version 1.25, October 2021

Arkouda:

- Bill Reus's CHIUW 2020 keynote talk: https://chapel-lang.org/CHIUW2020.html#keynote
- Arkouda GitHub repo and pointers to other resources: https://github.com/Bears-R-Us/arkouda

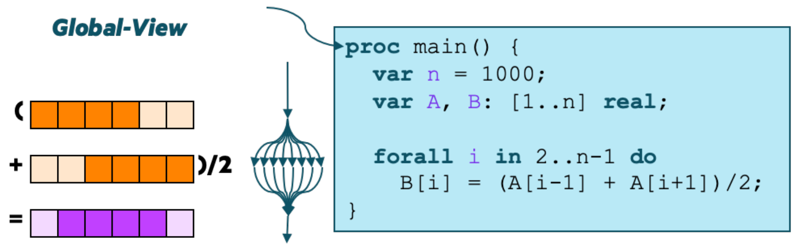
CHAMPS:

- Eric Laurendeau's CHIUW 2021 keynote talk: https://chapel-lang.org/CHIUW2021.html#keynote
  – two of his students also gave presentations at CHIUW 2021, also available from the URL above
- Another paper/presentation by his students at https://chapel-lang.org/papers.html (search "Laurendeau")

SUMMARY

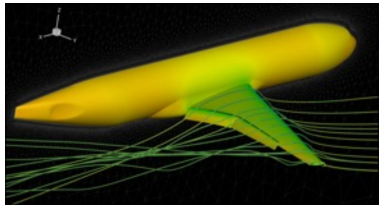
Chapel is unique among HPC programming models

its post-SPMD nature, lexical PGNS, and multiresolution philosophy
make HPC programming far more accessible
without throwing away control or performance

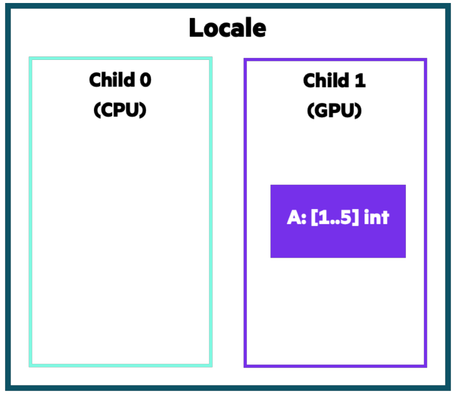


Chapel is being used for productive parallel programming at scale

- users are reaping its benefits in 19k–73k-line applications



Though significant challenges remain in targeting GPUs with Chapel, support is improving by leaps and bounds



We'd enjoy engaging with DOE employees who'd like to know more

THANK YOU

https://chapel-lang.org
@ChapelLanguage