A case for parallel-first languages in a post-serial, accelerated world

Paul Sathre
Research Software Engineer
Synergy Lab & NSF Center for Space, High-Performance and Resilient Computing
Virginia Tech
Disclaimers

• Imperfect tools are better than no tools, and doing serious science with them should be lauded, even as we strive to make better ones.
  – Critical feedback does not diminish the value that prior art has given us

• All thoughts and opinions expressed are my own, and shouldn’t be attributed to my employer, coworkers, or sponsors

• Logos and product names belong to their respective rights holders, and are neither an endorsement of this talk, nor of the referenced products
My Background → portable accelerated computing

• Undergrad:
  – Went in with a high-end desktop for “course work”
    • Quad-core 64-bit AMD, NVIDIA 8800 Ultra GPU, *Aegia PhysX physics processing unit* (PPU)
  – Roadrunner just coming online → friends & lab with PlayStation 3s
  – Brief stint doing bioinformatics on Tilera Tile64
  – GPGPU via OpenCL → two summers at LANL (cosmo + neutron transport)

• Grad school and shortly post-MS: CUDA ↔ OpenCL Interop
  – *MS Thesis*: CUDA to OpenCL translation → *(CU2CL)* on NVIDIA GTX 480
  – Interoperable MPI+{CUDA, OpenCL, OpenMP} for Micro-Air Vehicles → *(MetaMorph)*
    • Running on Nvidia GPU, AMD GPU, Intel MIC, respectively

• Since then: languages and tools for *modern* heterogeneous HPC
  – OpenCL support for FPGA: Linters *(FLOCL)* and autogenerators *(MetaCL)*
  – SYCL for irregular apps: AMD+NVIDIA GPU (via AdaptiveCpp), Intel FPGA (via DPC++)
  – GPU Chapel for irregular apps: perf./prod. tradeoffs → partitionability → portability (soon)
What I really care about:

Closing the gaps between the *parallel hardware* we already have, and *the people* who could benefit from it

So how do we *enable* them?
(Conversely, what are the *barriers* to use?)
Setlist

- Intro [you are here]
- Act I: Parallelism is everywhere, start acting like it
- Act II: The rise of GPUs, up through today
- Act III: Chapel’s role in our GPU future, and our role in Chapel’s
- Outro
ChapelCon #1: Looking back on ten CHIUW keynotes

- GPGPU was already in mind at CHIUW #1

- Python (and later Jupyter) interactive dev flows are important
  - *Keep turns within the human thought loop*, whether compilation or analysis

- Need a middle ground between FAANG-scale frameworks which often don’t scale *down* well, and laptop-scale which often don’t scale *up* well

- Analysis, Viz, Packaging, *community-alignment* are all important

- PGAS tends to beat explicitly-distributed when it comes to network-perf and productivity

- Flexibility and performance are *more important than transparency*
  - Start high level, but keep access and provide a smooth ramp to greater complexity
Act I

Parallelism is everywhere, start acting like it
Hardware is parallel, and likely to stay so

- Serial performance has barely improved since I started
- Parallel hardware was already common, now ubiquitous
  - Try to buy a laptop or cellphone without at least dual-core
- HPC has been using parallel and distributed software, but it’s still not very general
- Hardware is also heterogeneous, more on that in Act II
HPC moved to parallel, distributed, and heterogeneous long ago

But *nobody starts programming on an HPC cluster*
They start on a laptop/desktop

So what are *regular users* using?
What do desktop and laptop users actually have?

Steam hardware survey → rough proxy for general-purpose users
– In reach of {hobbyists, tinkerers, undergrads} → future HPC buyers

At least 68.17% CUDA-capable GPUs according to: https://developer.nvidia.com/cuda-gpus
At least 0.38% ROCm-capable GPUs according to: https://rocm.docs.amd.com/projects/install-on-linux/en/docs-6.1.1

Data collected 5/28/24 from https://store.steampowered.com/charts/
HPC is changing, **broadening**

- Users and developers are more mobile/remote
  - Less cubicle high-power workstation, more laptop / BYOD / tablet / cell
  - Are centrally-installed dev environments still the norm? Or more individual?

- Multicore + GPU at home in gaming / streaming / editing rigs
  - Everybody’s a Twitch, YouTube, insert-platform-here star

- It’s not just the privately-owned datacenter anymore → lots of IaaS
  - Renting cycles as needed vs. surpassable Big Iron cap-ex
  - Anyone with a credit card can buy GPU cycles for AI, crypto, etc.

- Need a **unified** approach to programming **all of it**
  - CPUs, **GPUs**, NICs, and whatever comes next
So regular users have (or can get) *parallel hardware*

But isn’t *parallelism* hard to understand?
The natural order of the world is *massively parallel*!

- Humans have innate *experiences* and *understanding of parallel processes*
  - Beehives $\rightarrow$ Scatter/Gather
  - School of fish, flock of sheep $\rightarrow$ single instruction (sheepdog), multiple thread (sheep)
  - Check-out lines $\rightarrow$ task parallelism and work stealing
  - Road networks $\rightarrow$ numerous parallel and sync constructs

Images CC-by-2.0 (cropped):

https://commons.wikimedia.org/w/index.php?curid=71040887

https://www.flickr.com/photos/16801915@N06/19022810042

https://www.flickr.com/photos/193316968@N06/52086734514
We have broad access to *parallel hardware*

Humans experience *parallel phenomena* in their day-to-day lives

Why does it still seem *so hard*?

Because we teach *serial first*, through the lens of old, *post-serial* norms!
Reality and hardware are parallel, so teach that

• Even “serial” threads **don’t run in isolation**
  – OS time-slicing, async IO, ISP/power outages

• Teach how to be safe and effective in a **parallel world**
  – New driver → **defensive driving**
  – New programmer → **thread-safety** and **fault-tolerance**

• Should be teaching async, concurrent, and parallel **to all, and earlier!**
  – **After** C/systems sequence, as an **elective** is **too late** → serial habits already anchored
  – To move earlier, need a language that is easy, **parallel-first**, and **general**

• PGAS is more approachable for learning **distributed**
  – Just some “further” cores/mem with more latency and failure modes
  – Don’t snail-mail a co-located co-worker! Use a whiteboard, post-its, Kanban, **lunch meeting**
Post-serial or “serial with sprinkles”

- Dominant programming models are still **post-serial**
  - “Sprinkles”: *optional* libraries, pragmas, language extensions

- Chapel presents a different option: **parallel-first**
  - A non-separable part of the *keywords*, *data abstractions*, and *semantics* of the language (i.e. *promotion*)

- Why **parallel-first** matters?
  - Philosophical: *realign* languages to hardware, *demystify* parallelism
  - Technical: better ground-up **parallel safety** built into its fabric
  - Technical: *no conflicts* between base and parallel sprinkle, *they grow together*

  - This OpenMP API specification references of the OpenMP specification are expected to result in unspecified behavior.
A vector is just a linear collection of things.

Should we express our code according to an individual element’s experience?

Or do we actually care about the collective?

This is closer to the mental model.
Intermezzo

• The world and modern hardware are parallel, *let’s start acting like it*
  – Use parallel-first languages, and *teach them to new users*
  – Move our mental model from “The Hero [thread]’s Journey”, to “shepherds of threads”

• But we also have to think about *heterogeneous parallelism*...
Act I: Parallelism is everywhere ...

Setlist

• Intro
• Act I: Parallelism is everywhere, start acting like it
• [you are here]
• Act II: The rise of GPUs, up through today
• Act III: Chapel’s role in our GPU future, and our role in Chapel’s
• Outro
Act II

The rise of GPUs, up through today
Hardware is **heterogeneous**

- **Heterogeneous** hardware is ubiquitous (and *has been*)
  - {floating-point-, graphics-, physics-, signal-, crypto-, tensor-, data-, vision-, reconfigurable-, ...} Processing Units
  - Some get married to the CPU as a SoC, some don’t
    - NVIDIA Grace Hopper, AMD MI300A APU, AMD Versal FPGA, Altera Agilex FPGA, etc.

**Performance Share by Accelerator Family**

- June 2006: Clearspeed (1.4%)
- June 2008: IBM Cell (8.6%)
- Nov. 2010: NVIDIA Fermi (12.7%)
- Nov. 2013: Intel Xeon Phi (17.7%)
- + NVIDIA Kepler (12.2%)
- Nov. 2018: NVIDIA Volta (23.7%)
- June 2022: AMD MI250 (30.2%)

**Act II: The rise of GPUs...**

**Accelerator du jour** keeps shifting vendor & language
Hardware is **heterogeneous**

- Accelerator *du jour* keeps shifting vendor & language

- **Portable** languages necessary to reach everything and *reduce rework*!

November 2023:

**70.1% GPU**

By vendor:

- **NVIDIA**: 36.3%
  - Ampere: 14.3%
  - Hopper: 12.9%
  - Volta: 8.2%

- **AMD**: 24.9%
  - MI250: 24.9%

- **Intel**: 9.0%
  - Max Data Center: 8.9%

- **No Coprocessor**: 28.8%

(Source: Top500.org, June 2004 – Nov 2023)
Timeline of today’s GPU languages*

“Dark Ages”
AMD Close-to-Metal, Sh, Brook/GPU/+, Cg, HLSL, other shaders


Official GPU Support
1.24 Low-level NVIDIA
1.27 Multi-locale NVIDIA
1.30 AMD via ROCm 4.X+
1.31 Multi-locale AMD

Heroic Efforts

2005 2010 2015 2020
2.0F 2.0C/++ 2.5

1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 12.5

Act II: The rise of GPUs...

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world" *scale approximate
What can we learn from CUDA?

**Consistency is king**
- Pick a model, stick to it, and iterate on it when needed
- Build *and keep* knowledge: docs, forums, code examples
  - Dead links are dead ends, insights lost to time

**Reach people *before they learn something else***
- University programs – free training and GPUs – were a smashing success

**Give away your tools**
- Easy to install and use *everywhere*

**Welcome the small fish, they are many and *some will grow bigger***
- New codes, users, buyers *start on laptops* before growing to datacenter

**Drive the conversation**
- Know where you excel, and *show it to people*
GPU languages have a pick 2 of 3 problem

Where we want to be

Performance

Portability

Productivity

OpenCL

AMD HIP SDK

NVIDIA CUDA

OpenMP

OpenACC

I haven’t gotten to do much portability work yet, just a few spot tests on AMD RDNA3 GPU.

Hope you saw the talks in Session #2!

Synergie.cs.vt.edu

ChapelCon’24 -- June 7, 2024

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"
**GPU Productivity Example: Vector Add**

- Either: **concise but implicit** or **explicit but verbose**

### OpenMP 4.0+

```c
#include <omp.h>

void vecAdd(float *A, float *B, float *C, int n)
{
    #pragma omp target teams distribute parallel for simd(map(to: A[0:n], B[0:n]) map(from: C[0:n]))
    for (int i=0; i<n; i++)
    {
        C[i] = A[i]+B[i];
    }
}
```

### CUDA

```c
__global__ void vecAddKernel(float *A, float *B, float *C, int n elem)
{
    size_t pid = blockIdx.x * blockDim.x + threadIdx.x;
    if (pid < n elem)
    {
        C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];
    }
}
```

### SYCL

```c
void vecAddKernel(__global float *A, float *B, float *C, int n elem)
{
    size_t pid = get_global_id(0);
    if (pid < n elem)
    {
    }
}
```

### OpenCL via MetaCL

```c
__kernel void vecAddKernel(__global float *A, float *B, float *C, int n elem)
{
    size_t pid = get_global_id(0);
    if (pid < n elem)
    {
    }
}
```

---

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"  
ChapelCon'24 -- June 7, 2024
Chapel GPU Productivity Example: Vector Add

- Chapel allows you to be both \textit{concise} and \textit{explicit}
  - Concise $\rightarrow$ \textit{Productivity}  
  - Explicit $\rightarrow$ \textit{Maintainability}

Chapel

1. use GPU;
2. proc vecAdd(A_h: [] real(32), B_h: [] real(32), C_h: [] real(32)) {
3. \hspace{1em} on here.gpus[0] {
4. \hspace{2em} var A: [A_h.domain] real(32) = A_h;  
5. \hspace{2em} var B: [B_h.domain] real(32) = B_h;  
6. \hspace{2em} var C: [C_h.domain] real(32) = noinit;  
7. \hspace{3em} C = A + B;  
8. \hspace{2em} C_h = C;  
9. \hspace{1em} }
10. }

Kernel boxed in green

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"  
ChapelCon'24 -- June 7, 2024
Chapel GPU on *Irregular* Apps

- **Transliteration** of CUDA Edge- and Vertex-centric graph analysis pipeline → **Jaccard Similarity**
  - See my CHIUW’23 talk for more detail

- Non-trivial: 3D kernels, atomics, random-access

- Most recent line counts: CUDA 1212 vs Chapel 599 (-51%!!)

Relative kernel performance was great!

- 18 graph inputs (see appendix): $|E| = \sim 3-500M$, **avg. density**: $\sim 2.1-160$
- Min: **0.51x**, Max: **1.98x**, Geo. Mean: **0.87x**
- Performance gap on the sparser inputs
- Performance parity ($\pm$ 5%) on the denser inputs
- Performance **gain** on a handful!

- Currently working on partitionable multi-GPU, multi-locale version
As a language for GPGPU kernels, Chapel is pretty good!

More productive, more maintainable, similar (and improving) performance

But how are people actually using GPGPU?
Is it post-serial GPGPU languages?

- CUDA clearly the market leader
- OpenMP and OpenCL on the decline
- ROCm and SYCL on the rise

Interest over time (Concept)

Google Trends Snapshot: 5/23/2024
Or is it *Python with Libraries*?

- High-performance Python → NumPy, et al
  - CuPy for GPU w/ compatible API is new, but growing
- TensorFlow & PyTorch compete w/ CUDA
- ONNX is following trends, but w/ lower share

Interest over time (Concept)

Google Trends Snapshot: 6/5/2024

---

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"
ChapelCon’24 -- June 7, 2024
Intermezzo

• GPUs are critical to modern performance, *both at exa- and laptop scales*

• Chapel’s GPU support is young but *productive, performant* and *portable*

• High-level Python has more *energy & interest* than low-level GPU
  – AI / Data science communities → new *federated* frontiers for HPC

• Chapel with GPU could *fill the gap* between *library-driven* Python and existing *post-serial* GPU languages!
Setlist

• Intro
• Act I: Parallelism is everywhere, start acting like it
• Act II: The rise of GPUs, up through today
• [you are here]
• Act III: Chapel’s role in our GPU future, and our role in Chapel’s
• Outro
Act III

Chapel’s role in our GPU future, and our role in Chapel’s
A very simplified view of AI & Data Science

Application Flow Logic

Bespoke Kernels

Analysis / Visualization

Where is multi-node?

“\textbf{I need more performance}” \rightarrow \textbf{jarring “leap of faith”}

Are the libs responsible for \textit{interop}, or am I?

Do I have to \textbf{move} data out of one lib and into another?

Will \textit{intra-} and \textit{inter-}node play nice?

pip install another library: (pyMPI, mpi4py, PySpark, DASK, ...)

High Performance GPU Library Kernels

\textbf{Too many [inter-]dependencies can lead to \textit{fragility} and \textit{overhead}}

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"
ChapelCon'24 -- June 7, 2024
Chapel eases path to parallel/GPU/distributed

Smooth ramp to complexity

Put intra- and inter-node into a language that *does both!*

Application Flow Logic

Bespoke CPU and GPU Kernels

NUMA-awareness / distribution via [co]locales

Analysis / Visualization

Wrapper APIs (new)

(Pre-existing) High Performance GPU Library Kernels

Retain interop to existing approaches

Do viz and analysis *where the data is being worked on!*

Looks a lot like Arkouda+ Arachne, and what we’ll hear about in Session #5

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world" ChapelCon'24 -- June 7, 2024
Available parallel and GPU Hardware ✓

Parallel-first, portable GPU Language ✓

AI and Data Science energy towards GPU ✓

So what is still in Chapel’s way?

Inertia ✗ Friction ✗ Exposure
Getting from A to B $\rightarrow$ $x_t = x_0 + tv_0 + \frac{1}{2}a_0 t^2$

- **Inertia (Position $\rightarrow$ language share)**
  - Old codes & langs are **battleships**: big, expensive, **powerful**, moved by committee
  - **Transliteration** is necessary to “prove it works” but high effort relative to reward

- **Friction (Velocity $\rightarrow$ rate of new codes)**
  - Programming is not just about the code, **it’s about the whole ecosystem!**
    - Installation, editor tooling, documentation, debugging, support forums, visualization, ...
  - Ecosystem ease of use can **help** or **hinder** adoption
  - Familiar workflows must be **trivial to reproduce**, or better yet, **improved upon**

- **Exposure (Acceleration $\rightarrow$ rate of change in rate of new codes)**
  - The **earlier** and **more broadly** you can reach people, the better
  - Users’ **struggles** will identify friction points, and **successes** will feed excitement loop!
What can we learn from CUDA?

Consistency is king → **Positive Inertia**

Reach people *before they learn something else* → **Pre-Inertia Exposure**

Give away your tools → **Exposure and Low Friction**

Welcome the small fish, they are many and *some will grow bigger* → **Exposure, Low Friction and Positive Inertia**

Drive the conversation → **Exposure**
How are we really doing on these?
Progress on Inertia

- We’re building on lessons from post-serial giants – Not going anywhere, but we grow as a complement to them
  - Higher-level for most, dip into post-serial via interop when needed

- Year-over-year we’ve increased breadth and depth of Chapel codes – Most successes are from new codes and users, not transliteration!

- Rigorous, at-scale successes are slowly chipping away

- I think it wins on clarity for maintainability, but I’m not sure there’s enough Chapel-native developers for a robust HR ecosystem (yet) – It’s not just the code, it’s the availability of people who know how to work on it

New codes and app. areas in Session #4!
Friction points

- **2.0 improved** stability, docs, diags. Work ongoing on LSP tools, debugging, interactivity
- **“Where .exe?”** Need better install, migration, *in situ* viz → where is our matplotlib?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP</td>
<td>Yes</td>
<td>Optional</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MPI</td>
<td>Yes</td>
<td>Advanced only</td>
<td>Advanced only</td>
<td>Yes</td>
</tr>
<tr>
<td>CUDA</td>
<td>Yes</td>
<td>NVIDIA required</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AMD HIP SDK</td>
<td>Yes w/ Repo</td>
<td>AMD or NVIDIA required</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>OpenCL</td>
<td>Yes</td>
<td>Optional</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYCL oneAPI</td>
<td>Yes w/ Repo</td>
<td>Optional</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYCL AdaptiveCpp</td>
<td>No</td>
<td>Optional</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CPU/single-loc</td>
<td>Soon!!</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>GPU/multi-loc</td>
<td>No</td>
<td>Optional AMD or NVIDIA</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Act III: Chapel's role in our GPU future...
What about Exposure?

• **We**'re here, so how’d we get here? → **Community survey** might show common themes

• **Curricula** are the long term answer, but **generality** is a precursor!
  – Right now part of **inertia**: slow battleships only moved by committees
  – But there are **other places to teach**: bootcamps, pre-college, online platforms!

• 3rd party blogs are **great** → independent credibility

• Increased social media needs to align to where prospective users **already are**
  – There’s a ton of CS, AI, Python, Rust, ProgrammerHumor on Reddit, but r/Chapel is quite thin

• Need to reach **beyond** HPC bastions, because **everyone** has parallel hardware now
  – Instead of “pushing out” from HPC spaces, “pull in” concepts and people from more general spaces

Check out Luca’s talk next in Session #3

Luca’s talk next in Session #3

Luca’s talk next in Session #3

Act III: Chapel's role in our GPU future ...
Where to put our efforts? → **Cascades** start *upstream*

- **✗ Inertia** is a giant / battleship, strong and slow
  - Small splashes won’t move it, but *riding a tide could → GPGPU!*

- **✓✓ Friction** is a solvable engineering problem
  - Make moving to Chapel *as easy as drifting downstream*
  - Programmers we’ve reached have *interest* and *energy* that we shouldn’t *waste on turbulence*
    - 2.0 stability, LSP tools, *installation*, interactivity, debugging, *visualization*

- **✓ Exposure** is important, but not a guarantee!
  - AI / data science devs are gravitating to more *general* languages
  - Reach into *general spaces* to show a *better path downstream*

- Focusing on the *upstream experience* and *potential energy* of newer users, could finally shift the tide to *parallel-first*
Conclusion

• The real world and everyday hardware are parallel, *so let's act like it*
  – Use *parallel-first* languages, like Chapel and *teach them to new users*

• Chapel’s GPU support learned quickly from prior GPU approaches
  – Now a *productive, performant, and portable* contender

• Let’s *align* to broader GPU trends towards a future in AI / Data science
  – *Smooth the gap* between high-level libraries, and low-level post-serial

• To grow a Cascade, seek *general* users & provide a *low-friction* experience
  – Reduce barriers to entry, expand reach from HPC to *general parallelism*
Thanks

• ... to current members and alums of Synergy Lab @ VT

• ... to the Chapel team at HPE for Q&A, feature requests, discussion
  – ... and inviting me to speak today!
  – Particular shoutouts to GPU and IO teams: Brad, Engin, Andy, Ben H., Michael, Lydia

• ... to the Chapel community for fighting hard on inertia, exposure, and friction

• Sponsorship
  – The work detailed herein has been supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).
Question & Answer

• The real world and everyday hardware are parallel, \textit{so lets act like it}
  – Use \textit{parallel-first} languages, like Chapel and \textit{teach them to new users}

• Chapel’s GPU support learned quickly from prior GPU approaches
  – Now a \textit{productive, performant, and portable} contender

• Let’s \textit{align} to broader GPU trends towards a future in AI / Data science
  – \textit{Smooth the gap} between high-level libraries, and low-level post-serial

• To grow a Cascade, seek \textit{general} users & provide a \textit{low-friction} experience
  – Reduce barriers to entry, expand reach from HPC to \textit{general parallelism}
Difficult Discussion Questions

• If a business has a Chapel-driven product or internal tool, how quickly can they bring a new grad up to speed to work on the language?

• If we use Chapel’s higher-level parallelism for teaching, are we hiding parts of the underlying parallelism that are critical for developing deep understanding?

• After improving base language installation, what is our package ecosystem like? Is / could Mason be as easy as Pip, Cargo, etc.?

• How well and portably do we expose GPU features like threadfence, tensor cores?

• Is a new language more or less rigid than coding to a [large] C/++ framework?
Wishlist

- Viz Viz Viz and Interactivity
- GPU-enabled install packages
  - WSL for students, deb for me, rpm and IaaS images for business
- Evangelism, show how good 2.0 is on real, important problems
- Tutorial series, in places where people are learning about code
  - YouTube, Twitch, Reddit, Blogs?
  - Installation all the way through to first GPGPU code
- Curricula examples for profs to grab-n-go
  - Auto-grading plugin(s) via LSP?
- Automatically co-schedule forall/foreach to CPU+GPU(s)
  - i.e. not an explicitly-partitioned cobegin
  - Ok to start with strictly unified memory within a single locale
- Intel GPU support eventually → 3 competitors is good for user pricing
Select artifacts

- **OpenDwarfs**: [https://github.com/vtsynergy/OpenDwarfs](https://github.com/vtsynergy/OpenDwarfs)
  - 13 computational idioms for OpenCL. Would love to see “Chapel-tastic” versions!
- **CU2CL**: [https://github.com/vtsynergy/CU2CL](https://github.com/vtsynergy/CU2CL)
- **MetaMorph**: [https://github.com/vtsynergy/MetaMorph](https://github.com/vtsynergy/MetaMorph)
- **MetaCL**: [https://github.com/vtsynergy/MetaMorph/tree/master/metamorph-generators/opencl](https://github.com/vtsynergy/MetaMorph/tree/master/metamorph-generators/opencl)
- **Unpartitioned Chapel Jaccard**
  - Code: [https://github.com/vtsynergy/Chapel-Examples](https://github.com/vtsynergy/Chapel-Examples)
- **Other lab code**: [https://github.com/vtsynergy](https://github.com/vtsynergy)
  - And papers: [https://synergy.cs.vt.edu/publications.php](https://synergy.cs.vt.edu/publications.php)
  - Points of contact: {sath6220, feng} at cs dot vt dot edu
We did run into some issues porting a 3D CUDA kernel

- GPU foralls are only 1D (for now)
  - Solution: Linearize loop range, then de-

- for-by loops do NOT GPU-ize (yet)
  - Problem: non-constant by clause can halt
  - Solution: Replace with while-count

- Accumulate via atomicAdd did NOT GPU-ize before Chapel 1.31
  - Solution: Call CUDA’s via extern C
  - Now has a gpuAtomic<Func> API

- Could incrementally validate as kernels were transparently mapped to CPU

See my talk at CHIUW’23 for more on this
# Edge- and Vertex-centric Jaccard: Data

<table>
<thead>
<tr>
<th>Sparsest/Largest</th>
<th>Europe_osm</th>
<th>Road_usa</th>
<th>Road-roadNet-CA</th>
<th>Road-roadNet-PA</th>
<th>Delaunay_n24</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kmer_A2a</strong></td>
<td>Protein k-mers</td>
<td>European roads</td>
<td>US roads</td>
<td>California roads</td>
<td>Pennsylvania roads</td>
</tr>
<tr>
<td>V: 171M</td>
<td>E: 361M</td>
<td>V: 50.9M</td>
<td>E: 108M</td>
<td>V: 1.96M</td>
<td>E: 5.52M</td>
</tr>
<tr>
<td>Avg: 2.11</td>
<td>Range: 39</td>
<td>Avg: 2.12</td>
<td>Range: 12</td>
<td>Avg: 2.41</td>
<td>Range: 8</td>
</tr>
<tr>
<td><strong>Soc-LiveJournal1</strong></td>
<td>Social network</td>
<td>Web page links</td>
<td>Voronoi differentials</td>
<td>Dielectric resonator</td>
<td>Large door</td>
</tr>
<tr>
<td>V: 4.85M</td>
<td>E: 85.7M</td>
<td>V: 3.57M</td>
<td>E: 84.8M</td>
<td>V: 1.96M</td>
<td>E: 9.64M</td>
</tr>
<tr>
<td>Avg: 17.7</td>
<td>Range: 20.3K</td>
<td>Avg: 23.8</td>
<td>Range: 188K</td>
<td>Avg: 38.2</td>
<td>Range: 134</td>
</tr>
<tr>
<td><strong>Stokes</strong></td>
<td>VLSI process sim.</td>
<td>Coauthorship</td>
<td>Social network</td>
<td>Costarring Actors</td>
<td>CFD of engine fan</td>
</tr>
<tr>
<td>V: 11.4M</td>
<td>E: 516M</td>
<td>V: 416K</td>
<td>E: 18.8M</td>
<td>V: 450K</td>
<td>E: 30.5M</td>
</tr>
<tr>
<td>Avg: 45.1</td>
<td>Range: 1728</td>
<td>Avg: 45.1</td>
<td>Range: 76</td>
<td>Avg: 56.4</td>
<td>Range: 3298</td>
</tr>
<tr>
<td><strong>Sc-msdoor</strong></td>
<td>Medium Door</td>
<td>Hollywood-2009</td>
<td>SVM</td>
<td>Hollywood-2009</td>
<td>Random Triangulations</td>
</tr>
<tr>
<td>V: 11.4M</td>
<td>E: 516M</td>
<td>V: 416K</td>
<td>E: 18.8M</td>
<td>V: 450K</td>
<td>E: 30.5M</td>
</tr>
<tr>
<td>Avg: 45.1</td>
<td>Range: 1728</td>
<td>Avg: 45.1</td>
<td>Range: 76</td>
<td>Avg: 56.4</td>
<td>Range: 3298</td>
</tr>
</tbody>
</table>

- **Sparsest/Largest**
  - Kmer_A2a
  - Europe_osm
  - Road_usa
  - Road-roadNet-CA
  - Road-roadNet-PA
  - Delaunay_n24

- **Smallest**
  - Europe_osm
  - Road_usa
  - Road-roadNet-CA
  - Road-roadNet-PA
  - Delaunay_n24

- **Densest**
  - Stokes
  - Sc-msdoor
  - Hollywood-2009

Graph data and images CC-BY-4.0 from the SparseSuite Matrix Collection (https://sparse.tamu.edu/).
Preprocessed CSR binary files: https://chrec.cs.vt.edu/SYCL-Jaccard/HPEC22-Data/index.html

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world" ChapelCon'24 -- June 7, 2024
Edge- and Vertex-centric Jaccard: Kernel performance vs. original CUDA on RTX 3090

Higher is better!

See my talk at CHIUW’23 for more on this

CPU: AMD Threadripper 3960X
CUDA: 11.6 / driver 510.108.03
GPU: Nvidia RTX 3090
Chapel: pre-1.31 (d7664c9d81)

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"
ChapelCon'24 -- June 7, 2024

Sathre, P. "A case for parallel-first languages in a post-serial, accelerated world"
{AI, Data, Domain} Scientists need visualization!

• Python/Jupyter drive an interactive code → viz → analysis loop
  – Need to support visualization within the human thought loop to attract these workflows
  – Chapel doesn’t have a plotting module, but could C interop fill the role? “How hard can it be?”

• Experiment with Chapel → C → C++ interop via matplotlib++
  – 1024x512, 5-point stencil Jacobi heat diffusion toy, after 10k iters →
    • ✓ 62 lines of Chapel application with 8 calls to library
      – Grid subsampled to 26x13 in a single line for slow gnuplot backend
    • x ... but 32 lines of manual Chapel→C binding
      – not great, but could be automated via c2chapel
    • x ... and 97 lines of manual C→C++ binding

• There is a gold mine of wrappable AI and Data Science libraries
  – ... but would need both Chapel & C++ experience, rough otherwise

Starting condition: 50°
Boundary conditions: S=100, E=0, N&W=50
1-thick ghost padding