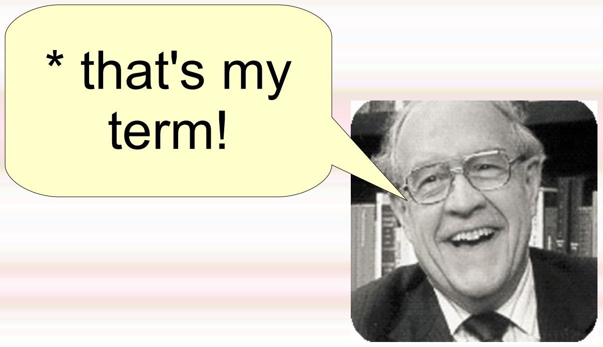
Finite Element Mesh Partitioning Using Perl+OpenMP

The Perl && Raku Conference
July 11-14, 2023
Toronto, CA Eh!

A Perl Boot Camp
About Perl Camp
imgflip.com

Goals for this Talk Are Not

- See pretty code
- See performance that scales as threads in-crease
- Present that OpenMP is a silver bullet*



Goals for this Talk Are

- Increase awareness of OpenMP in GCC
- Convince the Perl Community of the enormous opportunity embracing GCC's OpenMP represents
- Begin to identify key obstacles to more wide-spread awareness and adoption
- Present OpenMP as an available bullet* in Perl

Let's Define the Target User

C/OpenMP User, Perl dabbler

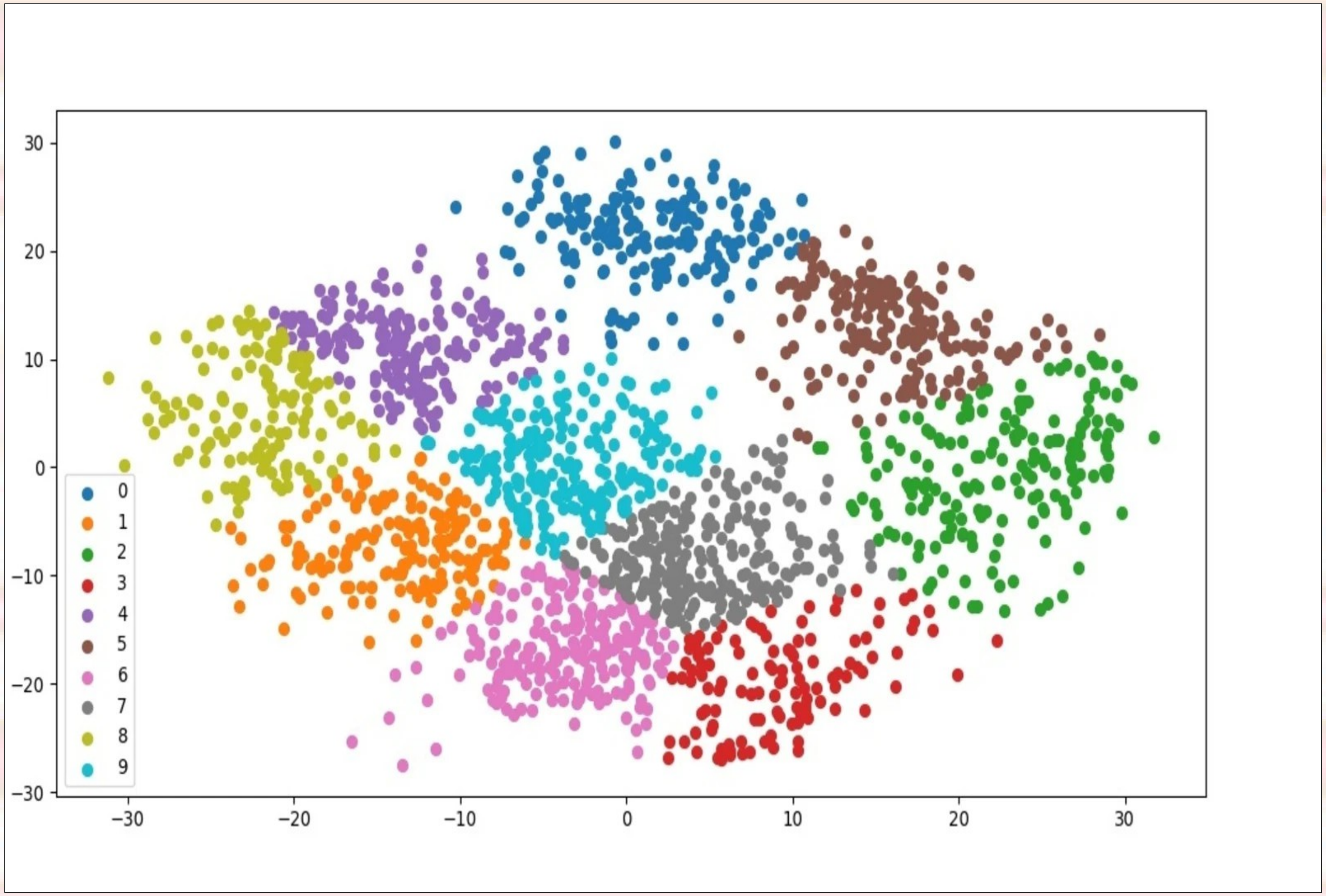
- Wants rapid prototyping of Perl
- Wants access to easy work-sharing offered by OpenMP
- Doesn't want to learn Perl API
- Probably has existing C/OpenMP code they wish to embed in a Perl driver
- Domain scientist (physical sciences, DMKD/ML)
- Not ready for PDL

Perl User, C dabbler

- Has already decide Perl is part of the solution
- Has tried other “thread” approaches in Perl, has found nothing in the *Goldilocks zone*
- Has been exposed to HPC or has supported HPC users
- Wants to process large datasets with lots of computational *locality*
- Has a machine with 8+ cores

DBSCAN

A data partitioning algorithm using a density function



https://dev.to/wmisingo/machine-learning-text-clustering-with-dbscan-399n

ADCIRC

Predicts hurricane storm Surge using an "unstructured" mesh (triangular elements)

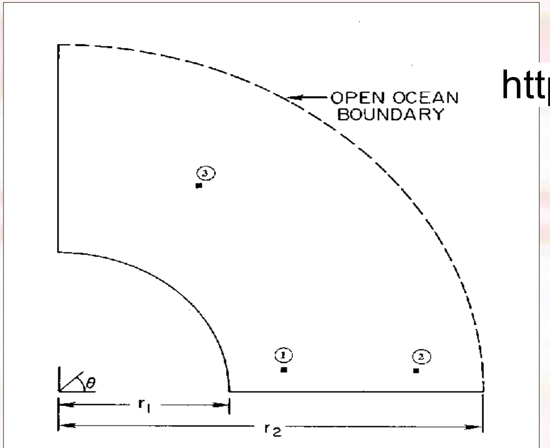


Figure E1.1 Geometry of the quarter annular test problem showing the location of the elevation and velocity stations (stations 1, 2, and 3)

https://adcirc.org

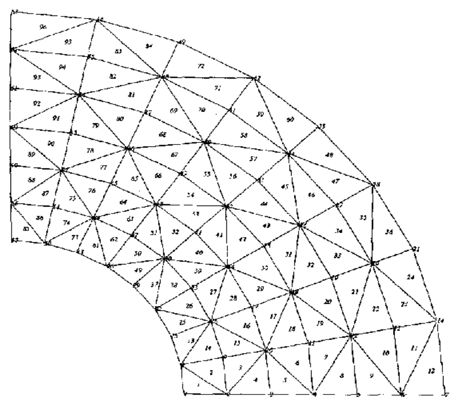
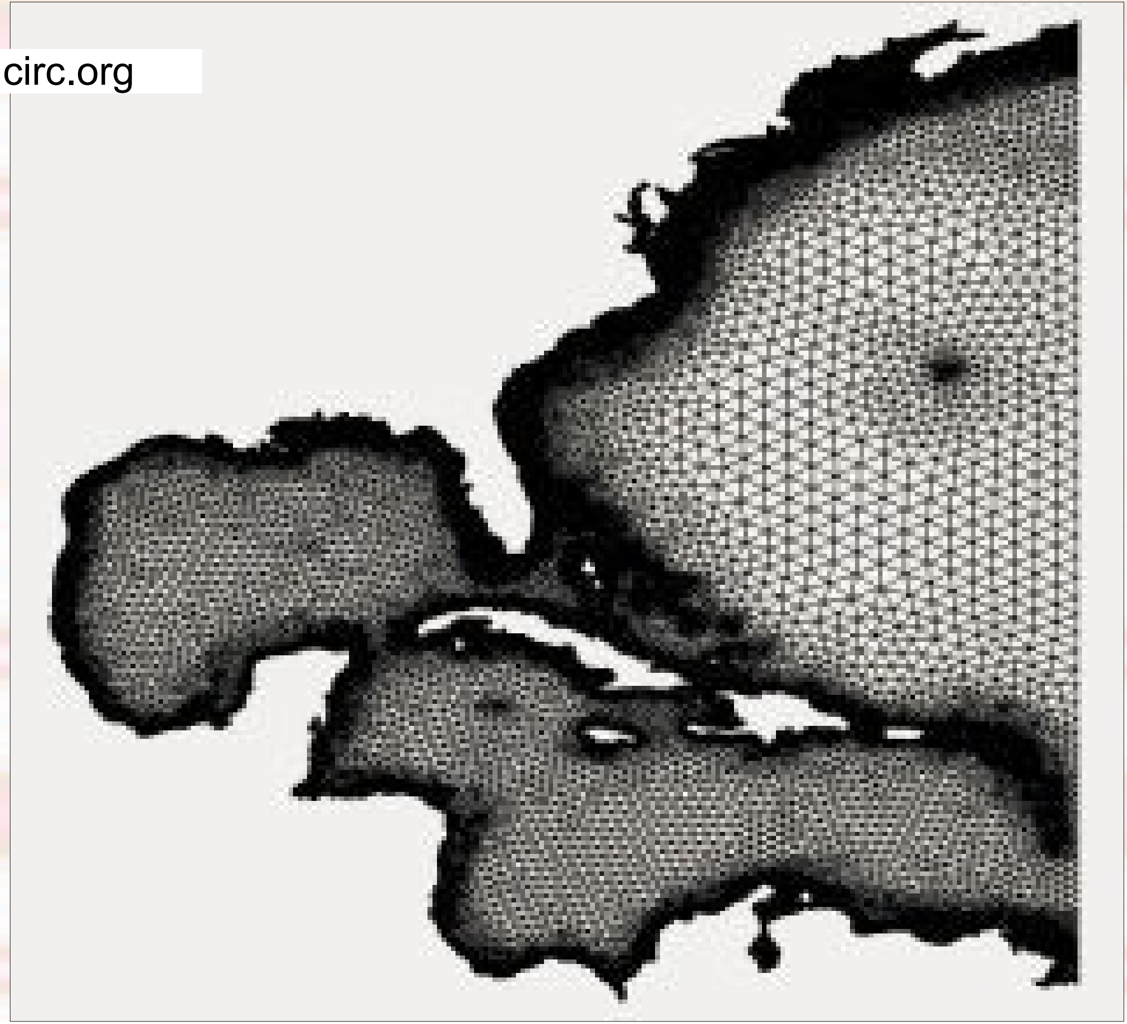


Figure E1.2 Finite element discretization of the quarter annular test problem showing node and element numbers

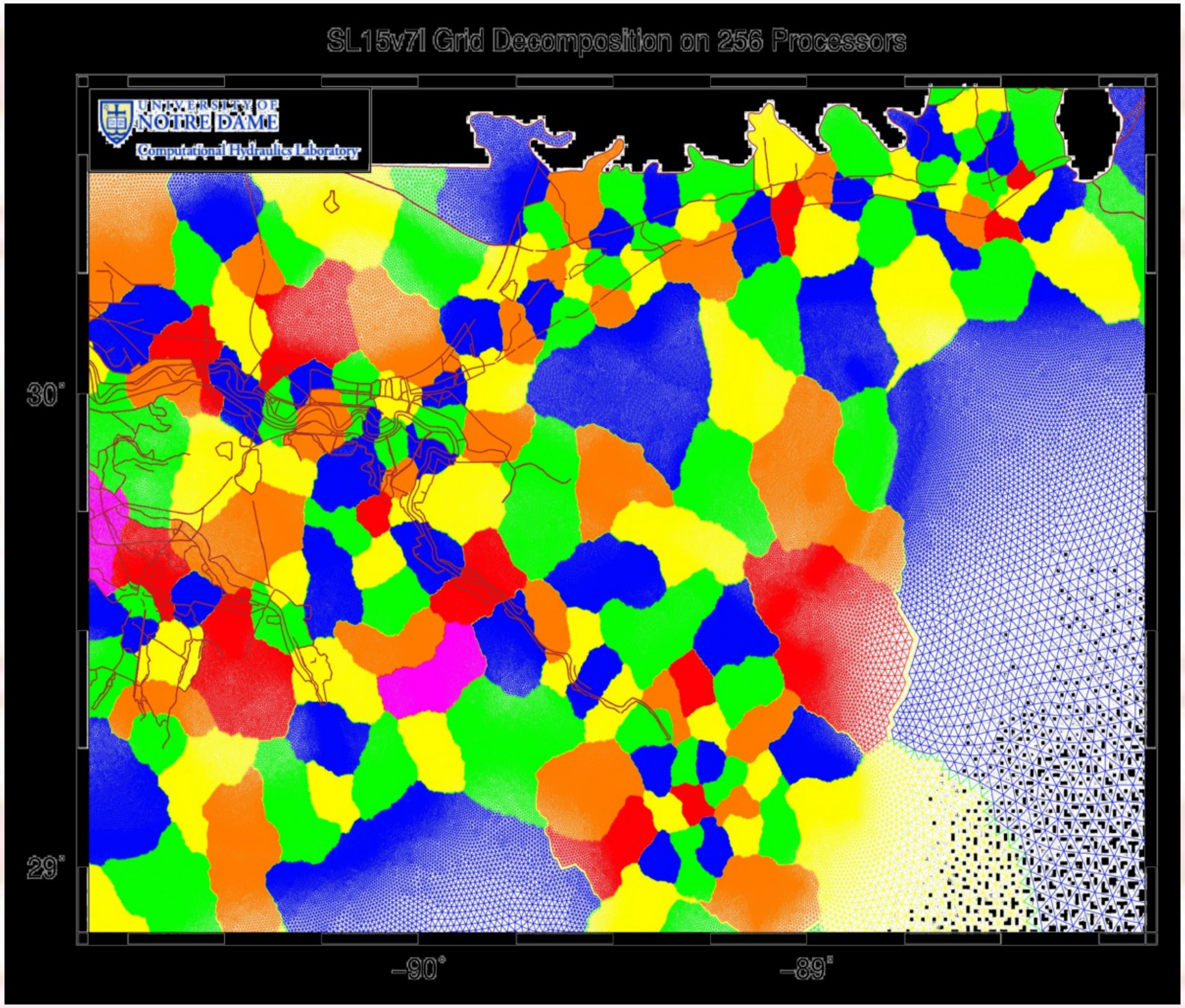


Parallel ADCIRC

Standard “domain decomposition” using METIS

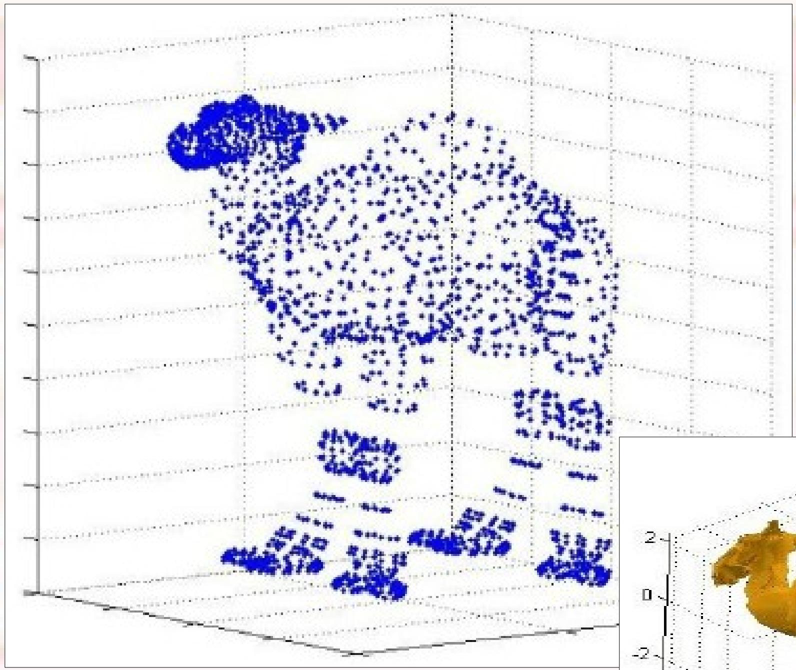
http://www.hpc.lsu.edu/docs/guides.php?system=QB2

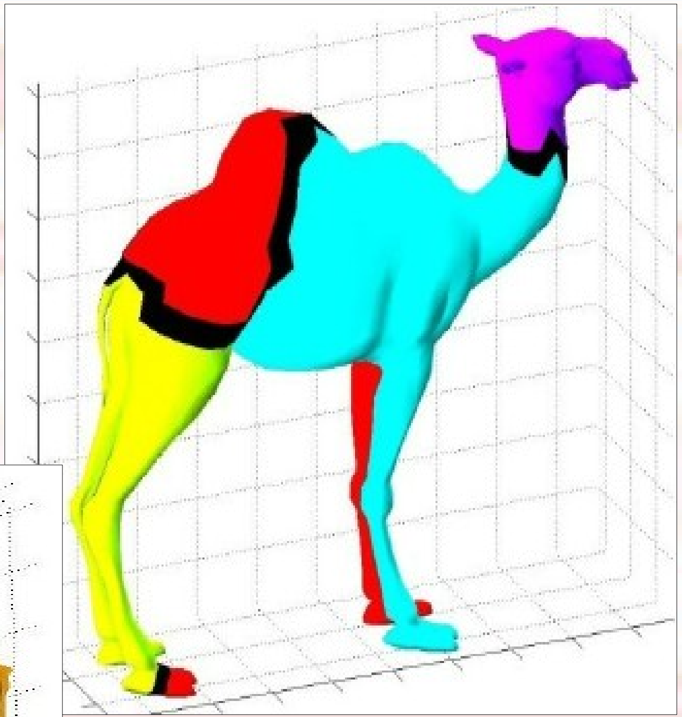
- ADCIRC meshes can be enormous
- Run in parallel on super computers
- Actually meshes are decomposed (partitioned into many smaller meshes)
- Standard method involves “METIS”, a library of algorithms aimed at mesh decomposition

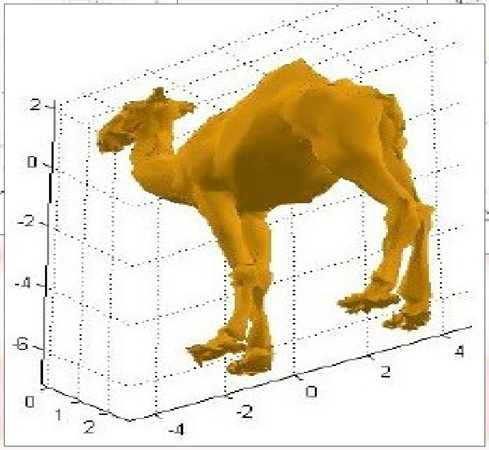

SL15v7l Grid Decomposition on 256 Processors
UNIVERSITY OF NOTRE DAME
Computational Hydraulics Laboratory
30°
29°
−90°
−89°

https://ccht.ccee.ncsu.edu/figuregen-v-49/

METIS Partitioning







https://www.researchgate.net/figure/3D-camel-model-under-different-attacks-a-additive-Gaussian-noise-b-Laplacian-mesh_fig3_261427900

Why Explore Alt. Domain Decomposition?

- ADCIRC is parallelized using a “message passing” technique (MPI)
- All touching subdomains must communicate over the network, network saturation is a know scaling limitation with ADCIRC
- METIS does not optimize for *communication complexity*, so there's interest in exploring others that do

Apply DBSCAN to a FE Mesh?

Pros

- ADCIRC meshes are most dense in areas of interest
- May result in net fewer “touching” subdomains
- Algoritmn is “simple” and can be optimized in several ways

(spoiler: using OpenMP)

Cons

- Also doesn't minimize communication complexity directly
- Computational complexity is $O(n^2)$
- A constant (ε) that defines points that are “close”
- It can be non-deterministic

Resulting Clusters

- … see images
- Results are in the right direction, but not as well defined … even accounting for the coarse application of ε
- HDBSCAN is a variation of DBSCAN that efficiently computes a solution using vari-able ε, and allows for the reconstruction of all the desired clusters

Code Inspection Time ...

(remember to open the code in the terminal)

Performance && Correctness

- Massive improvement moving from pure Perl to `Inline::C`
- No scaling improvements (*see* `NYTProf`) when using more threads, which is okay *for now*
- htop shows definitely that the threads are all working (show demo)
- Correctness of the clustering is suspect, more work to be done there also

An Examination of Code
The Good

- `Inline::C`, `Alien::OpenMP`, and `OpenMP::Environment` are all used
- Convenient macros were identified and *used* (e.g., `_POMP_ENV_UPDATE_NUM_THREADS_`)
- An appropriate execution module is being clarified
- The algorithm did kind of work as implemented, but some results are more muddled than I wanted
- I learned a lot about using the Perl API from the `Inline::C::Cookbook` (thank you Ingy, et al.!)

An Examination of Code
The Bad

- Expected scaling improvements were not achieved,
- e.g., I could not bare to move the outter loop in the Perl code into the C, which would have al-lowed the use of OpenMP's ability to "`collapse`" nested loops across threads when correctness was not violated (like in this case).
- Useful idioms were not able to be identified

An Examination of Code

The Ugly

- Perl data structures are wholly insufficient
- The required Perl API coding was way too low level
- The resulting code was neither Perlish nor at appropriately high level for most users of OpenMP

Confronting Ugly Pt. 1

The Perl API is Too Low Level for OpenMP

- Because Perl data types are internally so extremely uniprocess (which is actually a very powerful feature), and perhaps because of my own extreme knowledge gaps; I optioned to simply dump the data into basic C data types before any threading
- The Perl API code I wrote was very scary and very ugly

Confronting Ugly Pt. 2

Perl Needs Thread-safe Datatypes

- I had to use the Perl API code not only be-cause I could not safely modify an HV* or AV* across thread (which I verified as un-safe), but ...
- also because I was assured the even read-ing these structures by competing threads was not safe (I have not verified this, but I trust this is true)

The Beautiful Pt. 1

Higher Level of Perl API functions

- *For dumping 1d, 2d, 3d AVs of a uniform type (int, float) into C dimensionally equivalent arrays (OpenM-P's bread & butter!)*
- Easier *iterating over single and double leveled Hvs (even if values were constrained by C type)*
- *Way easier inspection of Hvs (keys to C array, sort-ing, and less tedious ways to "***`hv_fetch`***", get the value given the key*
- *Would be nice in as part of the core Perl API, but can likely be distributed with* **`Alien::OpenMP`**

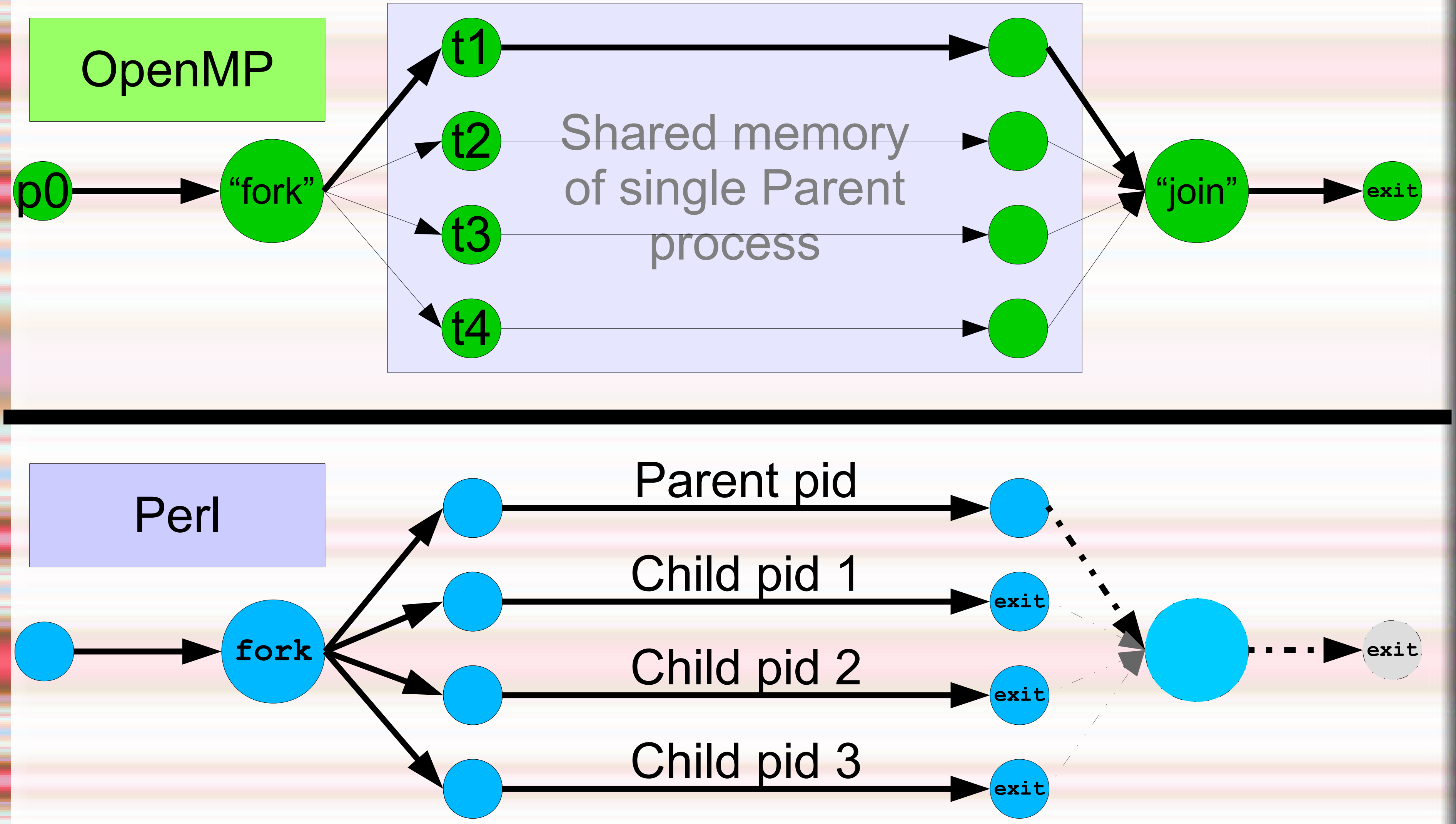
The Beautiful Pt. 2

Simplified, thread-safe Data structures in Core

- Lockfree data structures seems to fit extensively well here (skip atomics, at least for now)
- AVs, HVs that can be modified, even in a limited manner by competing threads, even if they exist in Perl API-land only
- Easy conversion between the lockfree data structures and current types

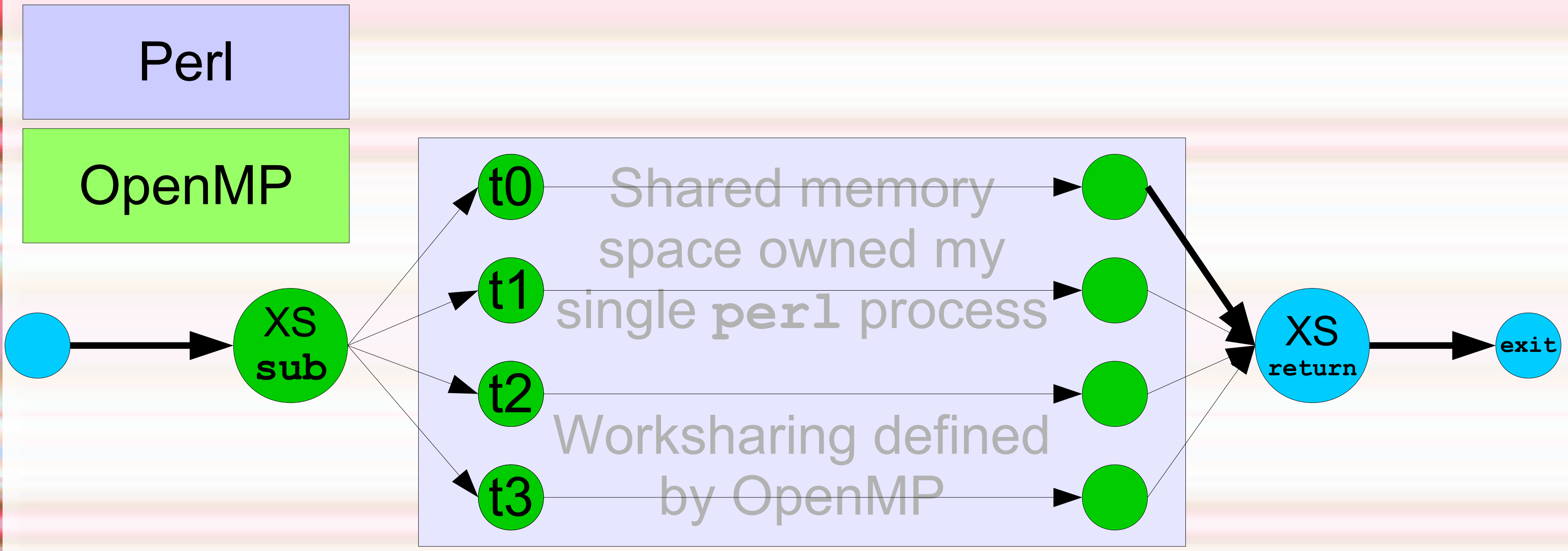
The Beautiful Pt. 3a

Perl and OpenMP Execution Models are Complementary ...



The Beautiful Pt. 3b

An Execution model Perl+OpenMP enables ...


Perl
OpenMP
XS sub
t0
t1
t2
t3
Shared memory space owned my single perl process
Worksharing defined by OpenMP
XS return
exit

The Beautiful Pt. 3c

Perl+OpenMP w/ Perl Datastructure Support

- Largely avoids the need to get HV* or AV* data into normal C containers
- Avoids use of atomic or critical sections, which in OpenMP; pipelining critical sections with independent Perl structures helps but doesn't solve the issue (simultaneous reads are also dangerous)
- Lockfree datastructures are well know and extremely relevant
- Minimal support of new types would go a long way!

Next Steps of Action

- Continuing working on the mesh decomposition tool using this method and others (e.g., k-part)
- Add a “helper” library **`#include`**, that contains useful MACROs (e.g., for `OMP_NUM_THREADS` envar) and “missing” Perl API functionality to **`Alien::OpenMP`**
- I will continue to force myself to learn Perl API, one day I am sure I will love it
- If I can implement any or all of the above, I will surely try
- We need to do something; help and feedback is wanted!
- TIMTOWTDI in full effect (let's do this!)

Thanks, Eh!