Threading in Pandora

Thoughts & First Look

Ryan Cross - LArSoft Threading Workshop

2023/03/03

WARWICK
THE UNIVERSITY OF WARWICK

Overview

This talk will cover:

1. **An overview of Pandora.**
2. **Pandora's current threading considerations.**
3. **A first look at threading work inside of Pandora.**
4. **Looking ahead at threading inside Pandora.**

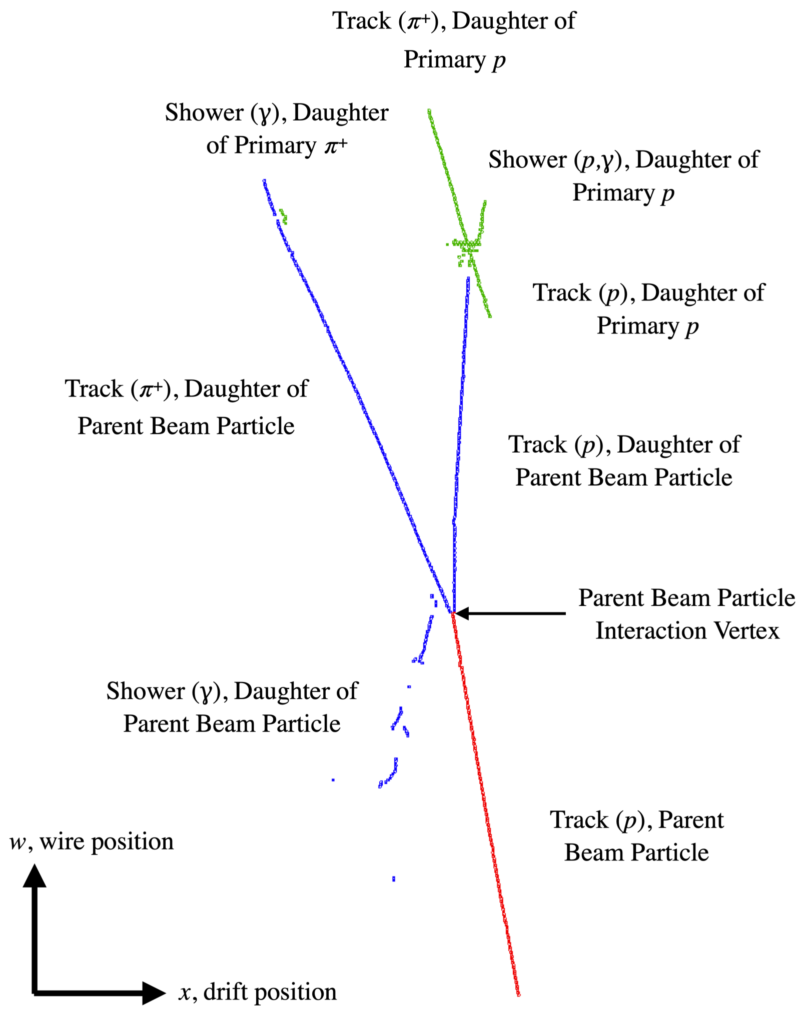
The Multi-Algorithm Approach to Pattern Recognition

Pandora is a software package developed for event reconstruction in high energy physics, in use across many LArTPC experiments. Pandora employs a **multi-algorithm approach** to pattern recognition, where a library of hundreds of algorithms are employed to gradually build up a full reconstruction of the event from the input hits.

These algorithms, from **traditional** to those which employ **deep learning**, are used to target small specific tasks such as:

- Clustering
- Vertex Creation
- 3D Hit Creation

The final output is a full 3D reconstruction of the event, with an overall particle hierarchy.

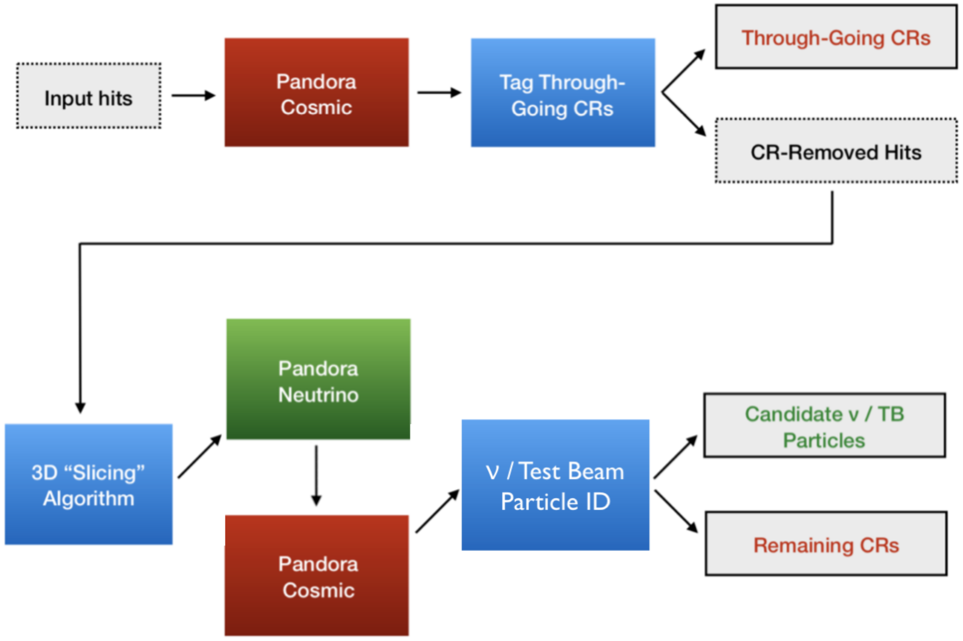


Track (π^+), Daughter of Primary p
Shower (γ), Daughter of Primary π^+
Shower (p,γ), Daughter of Primary p
Track (p), Daughter of Primary p
Track (π^+), Daughter of Parent Beam Particle
Track (p), Daughter of Parent Beam Particle
Parent Beam Particle Interaction Vertex
Shower (γ), Daughter of Parent Beam Particle
Track (p), Parent Beam Particle
w, wire position
x, drift position

PDSP Event from arXiv:2206.14521

Targeted Reconstruction Chains

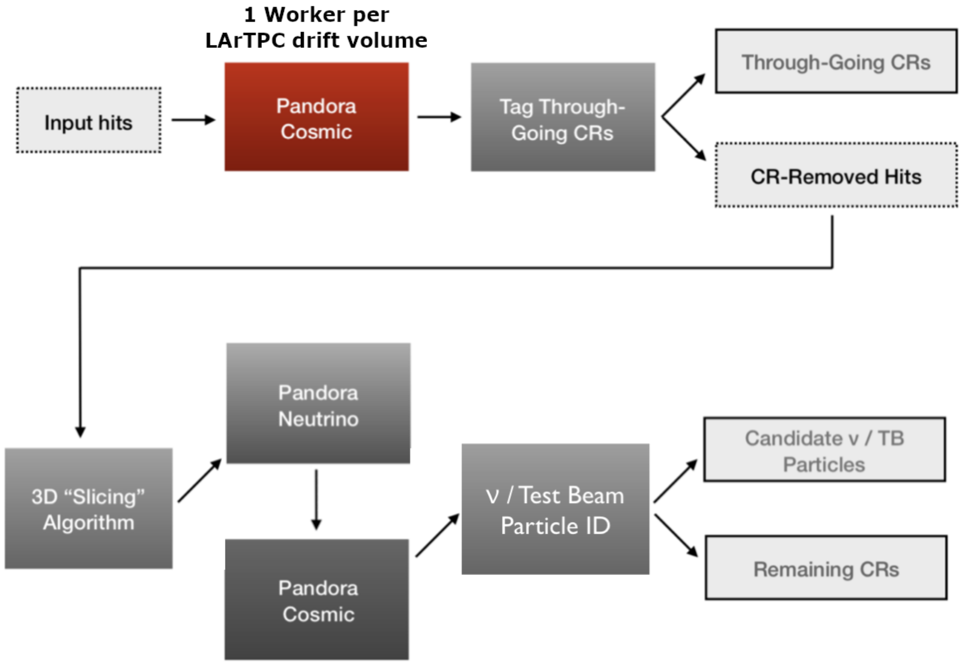
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interactions.

Targeted Reconstruction Chains

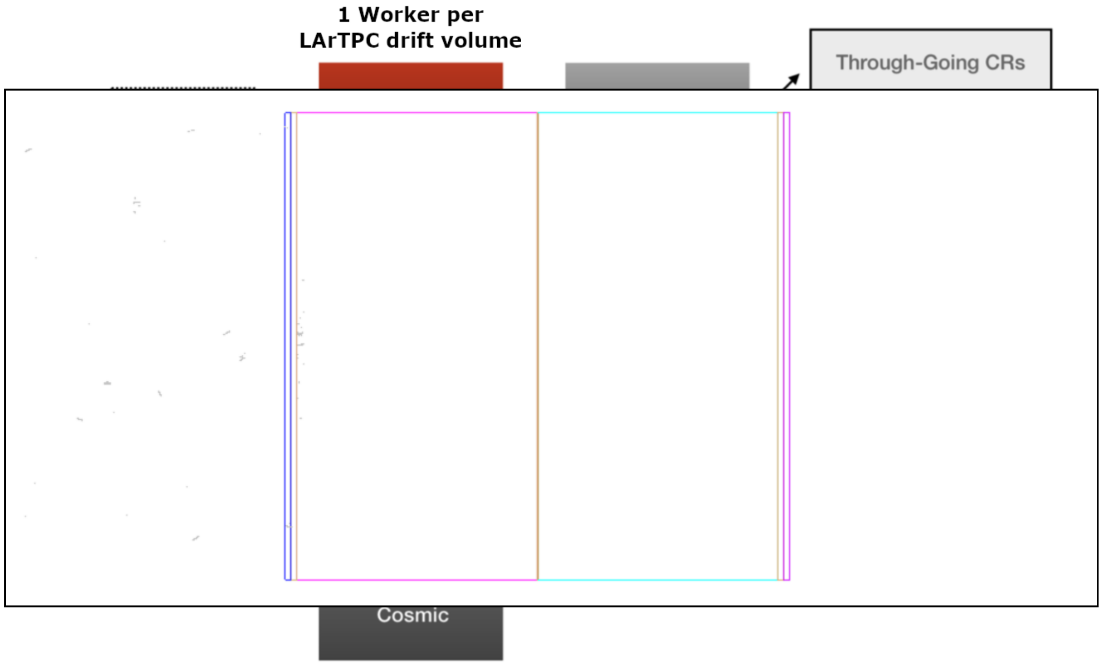
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interactions.

Targeted Reconstruction Chains

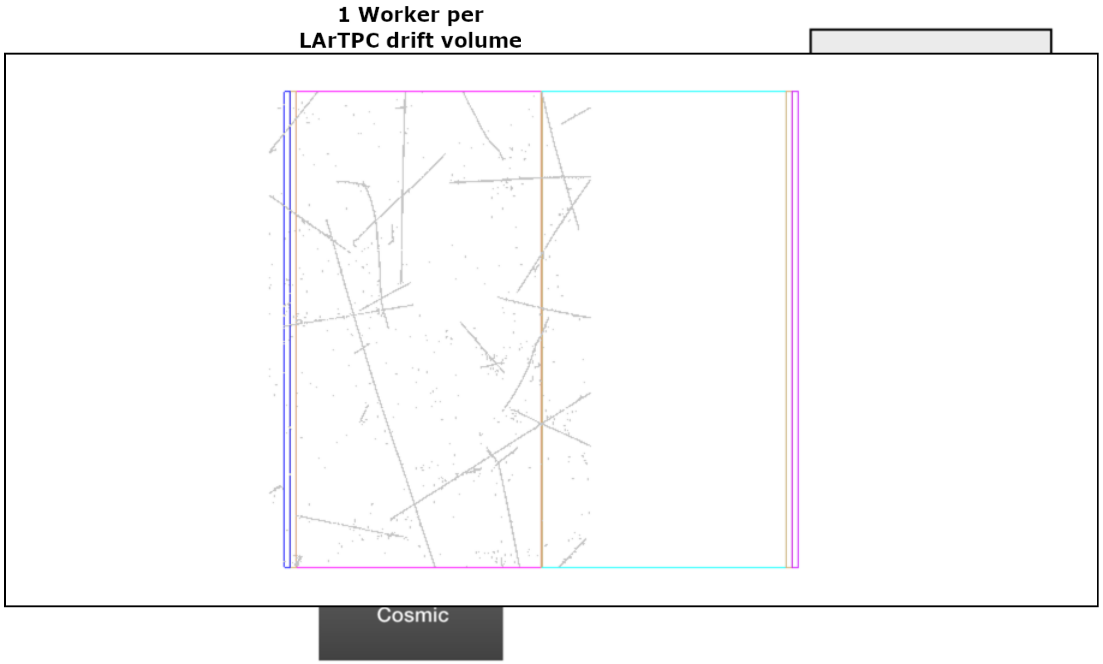
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interactions.

Targeted Reconstruction Chains

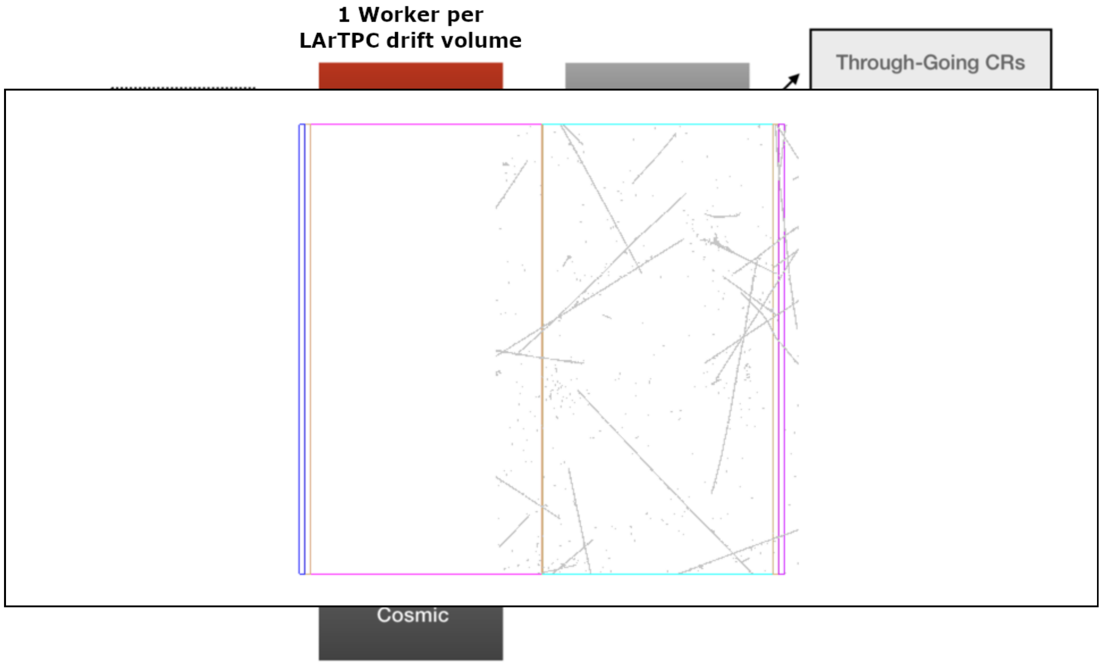
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interactions.

Targeted Reconstruction Chains

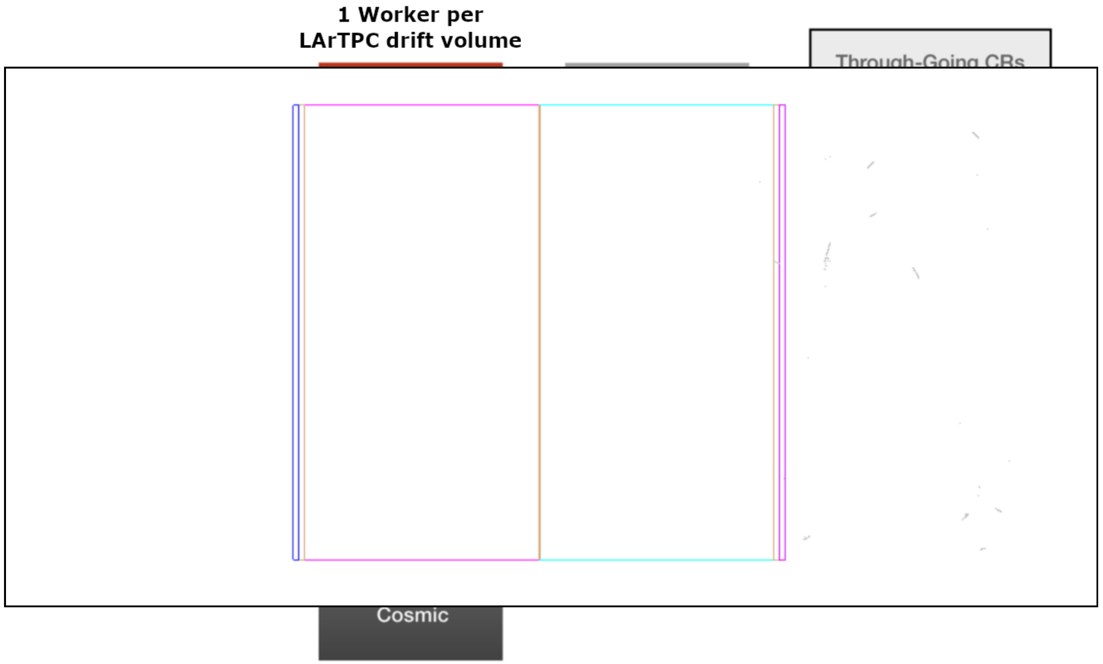
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interactions.

Targeted Reconstruction Chains

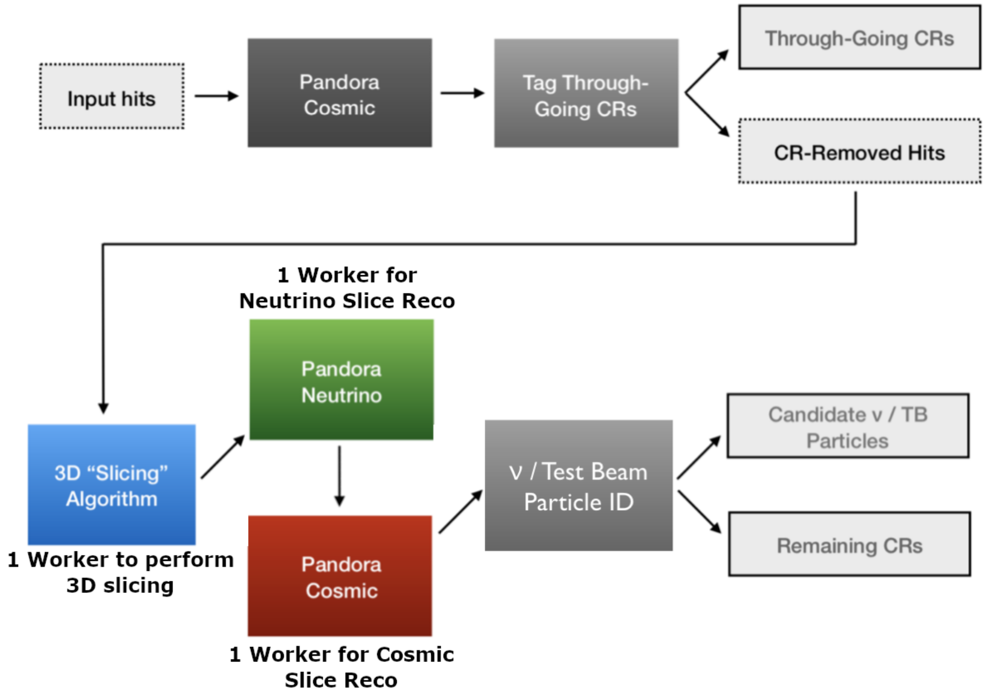
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interactions.

Targeted Reconstruction Chains

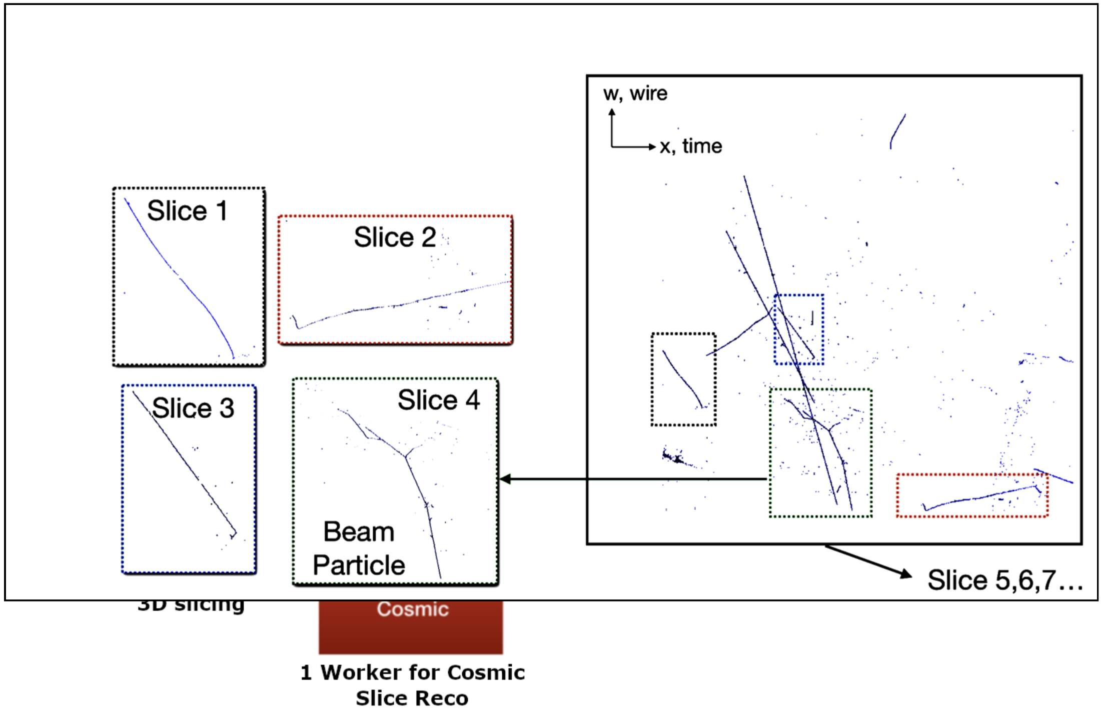
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.


Input hits
Pandora Cosmic
Tag Through-Going CRs
Through-Going CRs
CR-Removed Hits
1 Worker for Neutrino Slice Reco
Pandora Neutrino
3D "Slicing" Algorithm
1 Worker to perform 3D slicing
Pandora Cosmic
1 Worker for Cosmic Slice Reco
ν / Test Beam Particle ID
Candidate ν / TB Particles
Remaining CRs

This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interaction

Targeted Reconstruction Chains

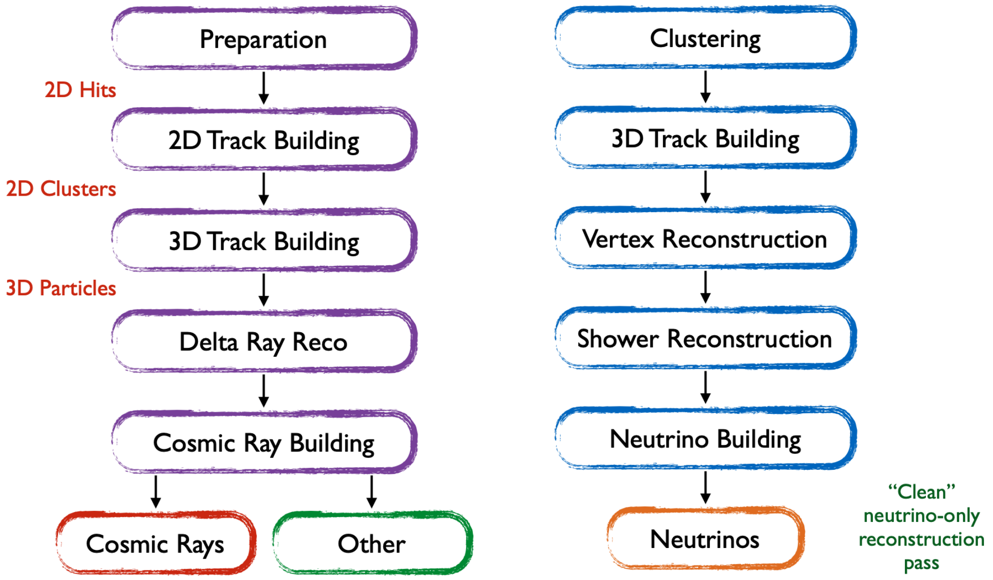
A strength of this multi-algorithm approach is the ability to having multiple targeted reconstruction chains. Pandora utilises these different algorithm chains to target different topologies.



This may be reconstructing both cosmic ray and test beam interactions in one instance of Pandora, or running different algorithm chains for interactions that have different reconstruction concerns such as atmospheric or supernovae interaction

Worker Configuration

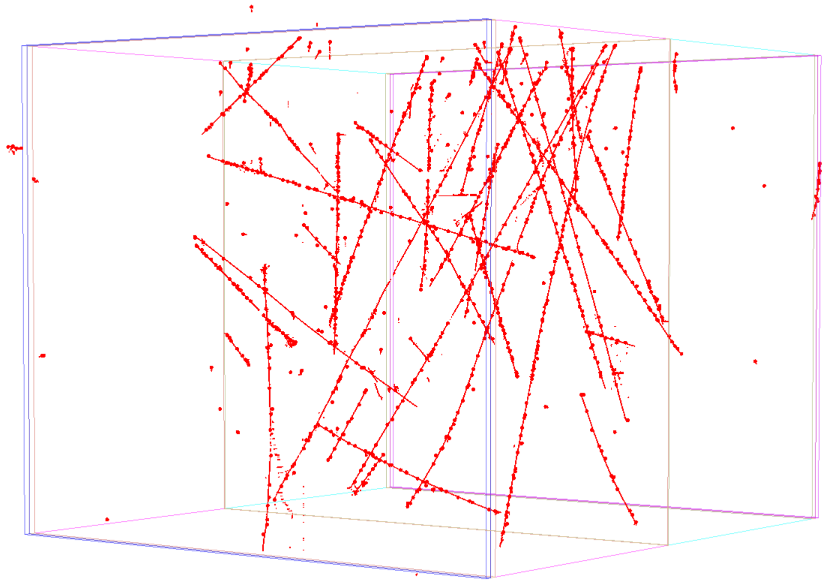
These workers are managed by a main Pandora instance, which controls how and when instances are launched, as well as deciding what to do with each output. This may be how it should be classified, or performing final reconstruction tasks to combine multiple outputs into one.

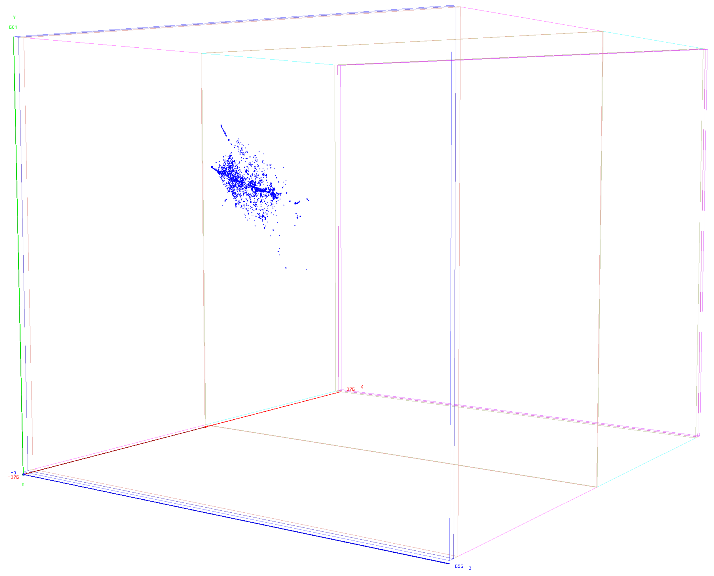


For example, a `Pandora Cosmic` and `Pandora Neutrino` instance can be used to most effectively target their respective interaction type, in a single global Pandora instance.

Worker Configuration

These workers are managed by a main Pandora instance, which controls how and when instances are launched, as well as deciding what to do with each output. This may be how it should be classified, or performing final reconstruction tasks to combine multiple outputs into one.





For example, a `Pandora Cosmic` and `Pandora Neutrino` instance can be used to most effectively target their respective interaction type, in a single global Pandora instance.

Where we are Now

Right now, Pandora does not utilise multi-threading anywhere*. This isn't ideal, especially in places where there is easy gains to be made by running multiple parts of the reconstruction workflow simultaneously.

However, threading has been considered as part of the development process of Pandora for many years. The individual workers that Pandora employs to target different target topologies have been written to be thread-safe, even if they currently do not utilise threads.

```cpp
#include "Api/PandoraApi.h"

#include "LArContent.h"
#include "LCContent.h"

#include "MicroBooNEPseudoLayerPlugin.h"
#include "MicroBooNETransformationPlugin.h"

int main(int argc, char *argv[])
{
    const pandora::Pandora *const pPandora1(new pandora::Pandora());
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::RegisterAlgorithms(*pPandora1));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::RegisterBasicPlugins(*pPandora1));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::SetLArPseudoLayerPlugin(*pPandora1, new lar_pandora::MicroBooNEPseudoLayerPlugin));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::SetLArTransformationPlugin(*pPandora1, new lar_pandora::MicroBooNETransformationPlugin));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ReadSettings(*pPandora1, "LArPandoraSettings.xml"));

    const pandora::Pandora *const pPandora2(new pandora::Pandora());
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LCContent::RegisterAlgorithms(*pPandora2));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LCContent::RegisterBasicPlugins(*pPandora2));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LCContent::RegisterBFieldPlugin(*pPandora2, bFieldParameters));
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ReadSettings(*pPandora2, "LCPandoraSettings.xml"));

    while (/* My Event Loop */)
    {
        PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ProcessEvent(*pPandora1));
        PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ProcessEvent(*pPandora2));

        // Extract output and persist

        PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::Reset(*pPandora1));
        PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::Reset(*pPandora2));
    }

    delete pPandora1;
    delete pPandora2;
}
```

LAr Pandora instance and Plugins

LC Pandora instance and Plugins

Loop over events, read-in within a Pandora Algorithm

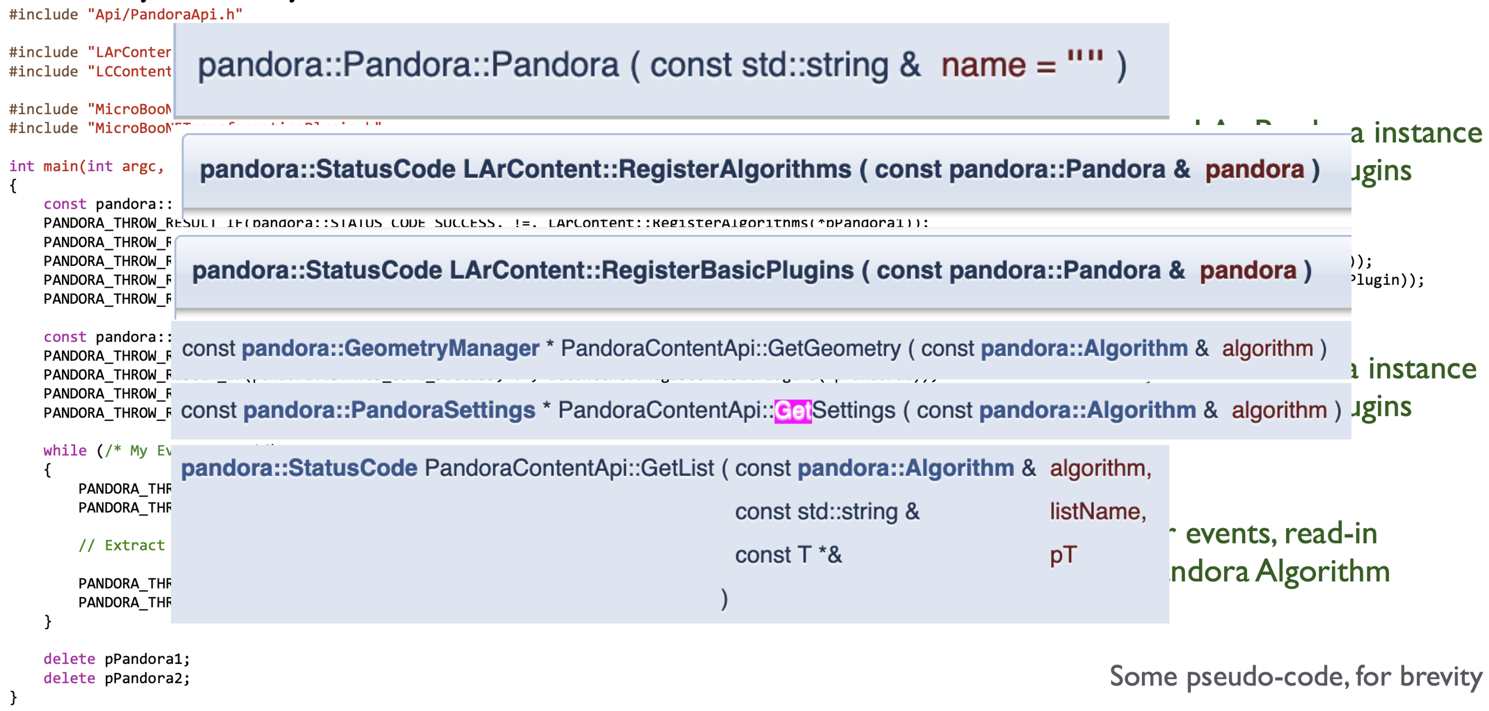
Some pseudo-code, for brevity

Pandora doesn't use threading in any code written by the Pandora development team. Dependencies such as ROOT, PyTorch, Eigen can of course use multi-threading.

Where we are Now

Right now, Pandora does not utilise multi-threading anywhere*. This isn't ideal, especially in places where there is easy gains to be made by running multiple parts of the reconstruction workflow simultaneously.

However, threading has been considered as part of the development process of Pandora for many years. The individual workers that Pandora employs to target different target topologies have been written to be thread-safe, even if they currently do not utilise threads.



```
#include "Api/PandoraApi.h"

#include "LArConter
#include "LCContent

#include "MicroBooN
#include "MicroBooN

int main(int argc,
{
    const pandora::
    PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::RegisterAlgorithms(*pPandora1));
    PANDORA_THROW_F
    PANDORA_THROW_F                                                                                          ));
    PANDORA_THROW_F                                                                                          Plugin));
    PANDORA_THROW_F

    const pandora::
    PANDORA_THROW_F
    PANDORA_THROW_F
    PANDORA_THROW_F
    PANDORA_THROW_F

    while (/* My Ev
    {
        PANDORA_THR
        PANDORA_THR

        // Extract

        PANDORA_THR
        PANDORA_THR
    }

    delete pPandora1;
    delete pPandora2;
}
```

pandora::Pandora::Pandora (const std::string & name = "")

pandora::StatusCode LArContent::RegisterAlgorithms (const pandora::Pandora & pandora)

pandora::StatusCode LArContent::RegisterBasicPlugins (const pandora::Pandora & pandora)

const pandora::GeometryManager * PandoraContentApi::GetGeometry (const pandora::Algorithm & algorithm)

const pandora::PandoraSettings * PandoraContentApi::GetSettings (const pandora::Algorithm & algorithm)

```
pandora::StatusCode PandoraContentApi::GetList ( const pandora::Algorithm &  algorithm,
                                                 const std::string &         listName,
                                                 const T *&                  pT
                                               )
```

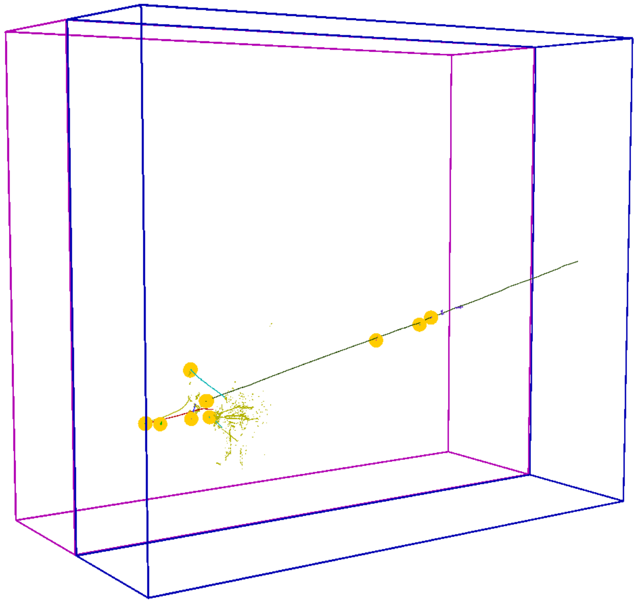
Some pseudo-code, for brevity

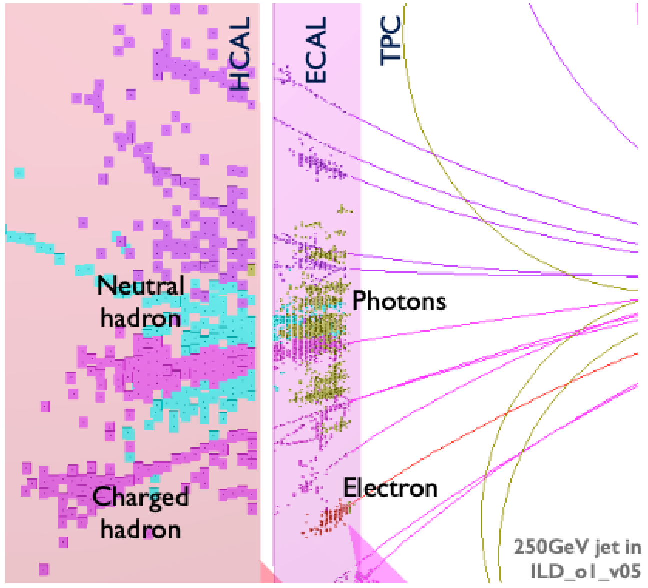
Pandora doesn't use threading in any code written by the Pandora development team. Dependencies such as ROOT, PyTorch, Eigen can of course use multi-threading.

Where we are Now

Right now, Pandora does not utilise multi-threading anywhere*. This isn't ideal, especially in places where there is easy gains to be made by running multiple parts of the reconstruction workflow simultaneously.

However, threading has been considered as part of the development process of Pandora for many years. The individual workers that Pandora employs to target different target topologies have been written to be thread-safe, even if they currently do not utilise threads.





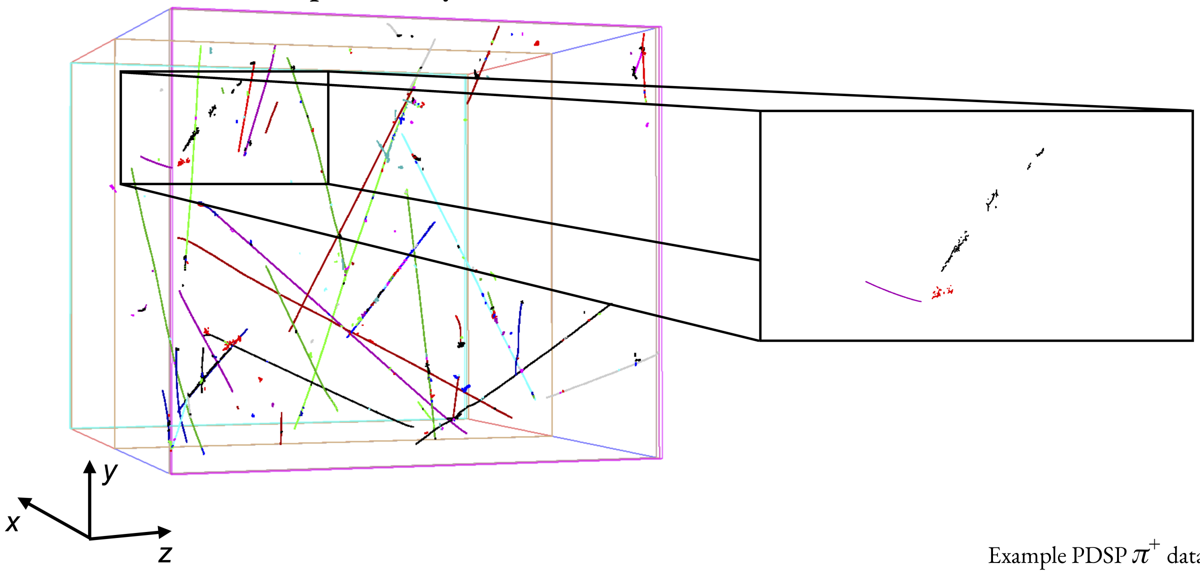
Pandora doesn't use threading in any code written by the Pandora development team. Dependencies such as ROOT, PyTorch, Eigen can of course use multi-threading.

Why Implement Threading?

Pandora is a key part of pretty much every LArTPC experiment, utilised heavily in the default workflows. This means that it is run extensively as part of the general `reco` steps or equivalent in pretty much every large scale production.

This means making changes to the performance should have an impact to many experiments, improving many workflows both production-based and interactive jobs.

However, it is also useful to be realistic about the total impact of Pandora, over the full generation, simulation and reconstruction chain at PDSP: it takes up only **3.8%** of the runtime on average over the full chain, or **17.8%** for the reco chain specifically.



Example PDSP π^+ data event from arXiv:2206.14521

How to Exploit Threading in Pandora

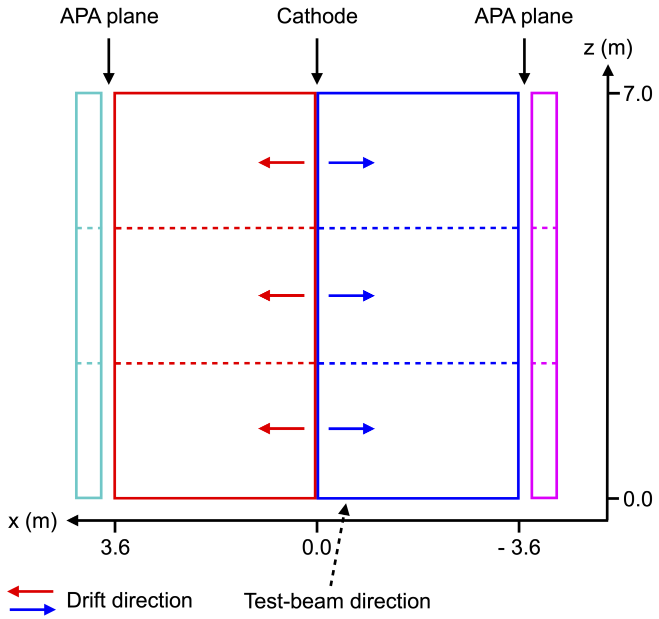
There is a few obvious places to utilise threading in Pandora, which we can use to start increasing the efficiency of Pandora:

How to Exploit Threading in Pandora

There is a few obvious places to utilise threading in Pandora, which we can use to start increasing the efficiency of Pandora:

Easy: Parallelise the Pandora workers we use:

- Cosmic reconstruction is very time consuming, but we can run each detector volume independently, for a decent speed up.
- Similarly, we can run multiple slices and slice hypotheses at the same time.



PDSP Figure from arXiv:2206.14521

How to Exploit Threading in Pandora

There is a few obvious places to utilise threading in Pandora, which we can use to start increasing the efficiency of Pandora:

Easy: Parallelise the Pandora workers we use:

- Cosmic reconstruction is very time consuming, but we can run each detector volume independently, for a decent speed up.
- Similarly, we can run multiple slices and slice hypotheses at the same time.

Harder: Allow the reco chains to use available threads as well:

- Rather than running many chains/workers in parallel, allow the workers to also utilise threads.
- Requires more thought to ensure only available resources are used, without slow down.

How to Exploit Threading in Pandora

There is a few obvious places to utilise threading in Pandora, which we can use to start increasing the efficiency of Pandora:

Easy: Parallelise the Pandora workers we use:

- Cosmic reconstruction is very time consuming, but we can run each detector volume independently, for a decent speed up.
- Similarly, we can run multiple slices and slice hypotheses at the same time.

Harder: Allow the reco chains to use available threads as well:

- Rather than running many chains/workers in parallel, allow the workers to also utilise threads.
- Requires more thought to ensure only available resources are used, without slow down.

Hardest: Start to move the full algorithm chain to allow asynchronous running:

- Certain algorithms aren't needed straight away, but can run on lower level information that is available early on.
- Could run some of the DL work in the background on the raw hits, and then utilise its output later on when needed, rather than running in sequence.

A First Look: What has been Done

As the first point is very simple to implement, at least for the cosmic ray reconstruction, this has been (naively) implemented into Pandora as a proof-of-concept.

Pandora produces a cosmic ray worker instance per detector volume, and these instances are run entirely independently of each other, making the introduction of parallelism here trivial. Once completed, the results are stitched together if needed and reconstruction continues.

Implementing threading here is as simple as having each thread take a pending worker instance from a queue, and running it to completion, and going back to the queue until all workers have been exhausted.

A First Look: What has been Done

As the first point is very simple to implement, at least for the cosmic ray reconstruction, this has been (naively) implemented into Pandora as a proof-of-concept.

Pandora produces a cosmic ray worker instance per detector volume, and these instances are run entirely independently of each other, making the introduction of parallelism here trivial. Once completed, the results are stitched together if needed and reconstruction continues.

Implementing threading here is as simple as having each thread take a pending worker instance from a queue, and running it to completion, and going back to the queue until all workers have been exhausted.

For a small bit of work, this has the potential for big gains, as cosmic ray reconstruction can be a large part of the overall reconstruction time inside Pandora:

Experiment	Total Time	CR Reco Time	Percentage
ProtoDUNE Single Phase	9h 18 m 48 s	6h 37 m 42 s	71%

These numbers are from Pandora standalone, running in a different environment to the LArSoft-based numbers, so aren't directly comparable.

A First Look: Performance Improvements at PDSP

With this simple improvement added, the cosmic ray reconstruction is improved by a decent amount, especially relative to ease of implementation of threading here.

A First Look: Performance Improvements at PDSP

With this simple improvement added, the cosmic ray reconstruction is improved by a decent amount, especially relative to ease of implementation of threading here.

Focusing on cosmic ray inference only (i.e. running only that step), the performance difference over 500 PDSP events is:

Experiment	**Reco Time**	**Time Per Event**
Without Threading	6 h 37 m 42 s	47.7 seconds
With Threading	5 h 10 m 19 s	37.3 seconds

These numbers are from Pandora standalone, running in a different environment to the LArSoft-based numbers, so aren't directly comparable.

A First Look: Performance Improvements at PDSP

With this simple improvement added, the cosmic ray reconstruction is improved by a decent amount, especially relative to ease of implementation of threading here.

Focusing on cosmic ray inference only (i.e. running only that step), the performance difference over 500 PDSP events is:

Experiment	**Reco Time**	**Time Per Event**
Without Threading	6 h 37 m 42 s	47.7 seconds
With Threading	5 h 10 m 19 s	37.3 seconds

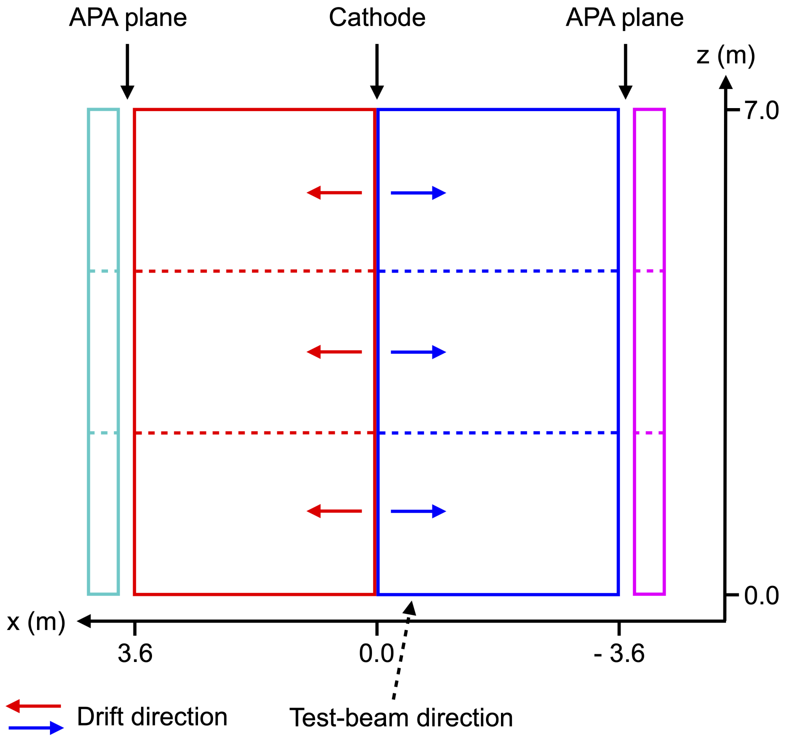
Which is encouraging! For a small change (~20 lines) it unlocks a decent performance uplift. However, it perhaps isn't as much as it would seem when the test had access to 4 threads.

These numbers are from Pandora standalone, running in a different environment to the LArSoft-based numbers, so aren't directly comparable.

Limitations

As an early, and quick implementation this is encouraging results. However, this form of parallelisation has a few main issues:

Not all drift volumes are equal. For example, in ProtoDUNE-SP the "4" drift volumes are really 2 larger and 2 small ones, with the small ones completing processing almost instantly, making the parallelising less impactful than it may first appear.

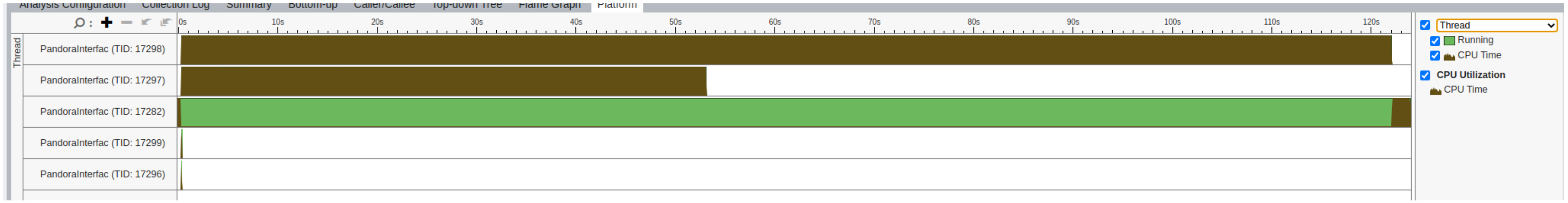


PDSP Figure from arXiv:2206.14521

Limitations

As an early, and quick implementation this is encouraging results. However, this form of parallelisation has a few main issues:

This form of parallelisation only enables multiple workers at once, rather than opportunistic usage of threads inside the workers themselves. It could likely be even more performant to parallelise some of the slower running algorithms as well as the workers themselves.



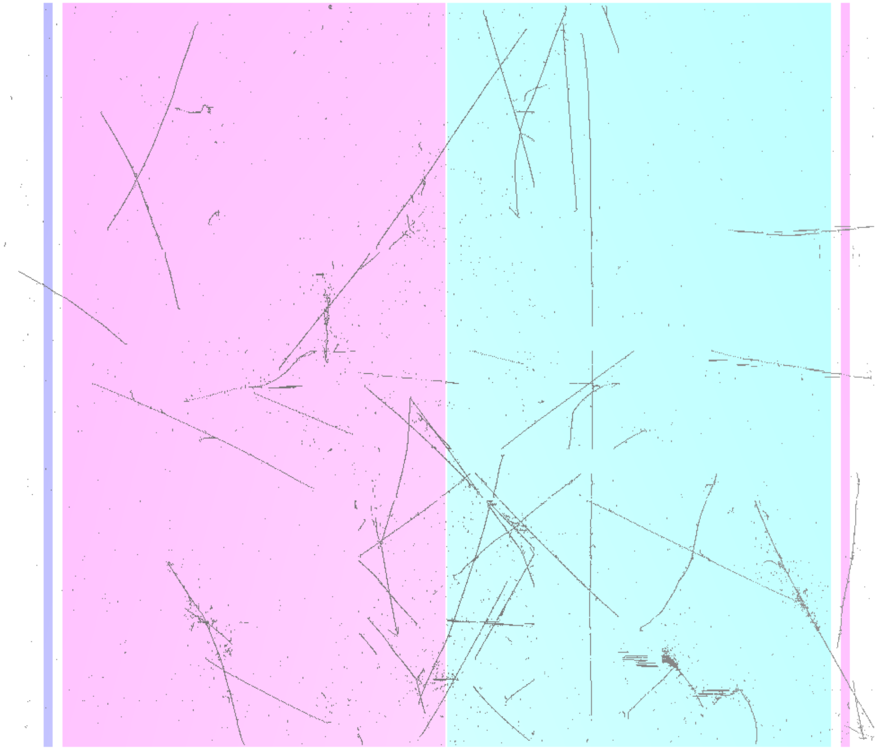
This shows the same issue as the previous slide, with 2 threads instantly completing, 1 lasting about 55 seconds, and the final thread taking just over 120 seconds. If the workers were set up to utilise threads as well, the system's resources would be utilised more efficiently, as the longer running workers could pick up the remaining resources as available.

Intel vTune

Limitations

As an early, and quick implementation this is encouraging results. However, this form of parallelisation has a few main issues:

This form of parallelisation only enables multiple workers at once, rather than opportunistic usage of threads inside the workers themselves. It could likely be even more performant to parallelise some of the slower running algorithms as well as the workers themselves.

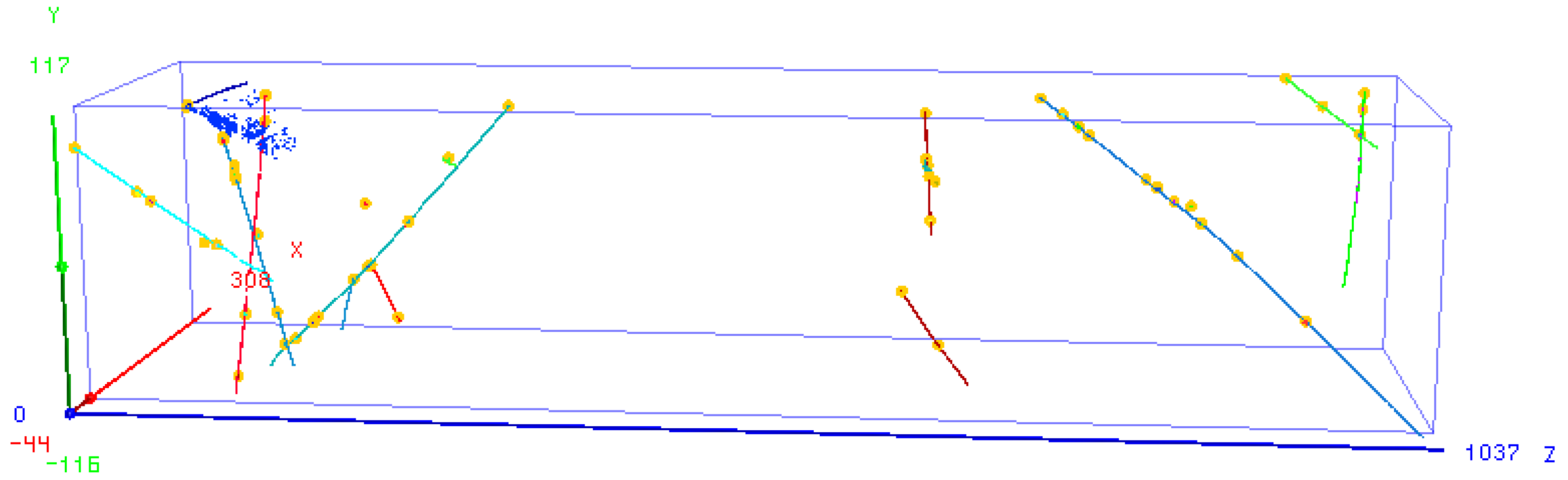


Intel vTune

Limitations

As an early, and quick implementation this is encouraging results. However, this form of parallelisation has a few main issues:

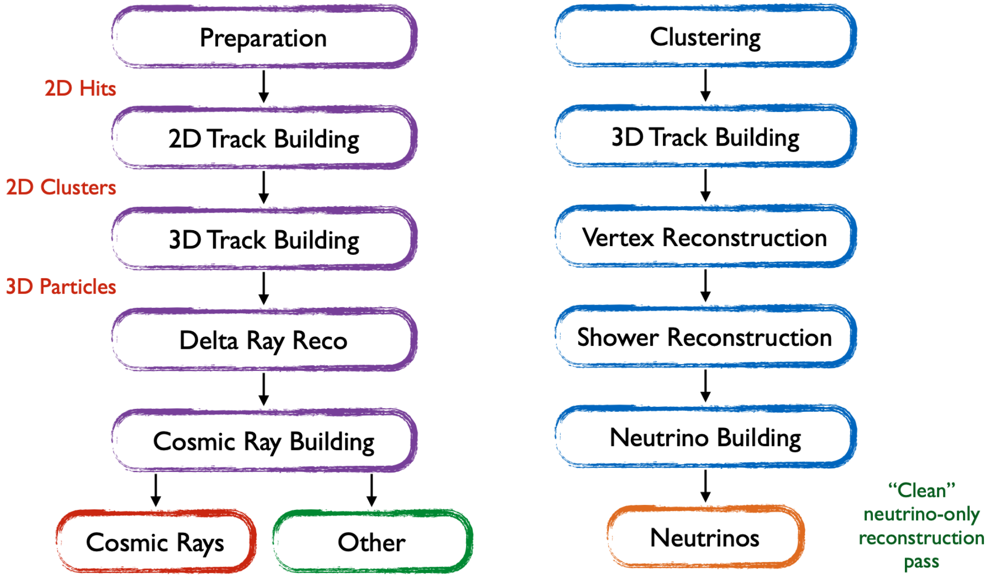
Finally, there is zero benefit in cases where there is only a single drift volume, such as MicroBooNE, or at experiments where the cosmic workers are not needed, as there is no workers to parallelise.



Limitations

These limitations, for the most part, are not unique to paralleling the cosmic workers, with similar issues for the slice reconstruction, once that is implemented, with the additional issue that the slice reconstruction is a smaller part of the overall reco chain.

What would help more, for both sets of workers and across every LArTPC type, is the ability to more easily parallelise the reconstruction chains themselves. For example, there are many parts of the reconstruction that run across large combinations of 2D/3D clusters, such that parallelising those individual algorithms and tools may prove more useful, as well as unlocking performance gains at detectors without cosmic rays or the need for slicing.



Next Steps

The next steps seem fairly obvious, though vary wildly in terms of difficulty. The obvious and not too difficult first steps are:

- Tidy up and double check the parallelised cosmic worker code.
- Implement a similar workflow for processing multiple slices.
- ~~Hook up a full end-to-end chain to propagate any threading parameters and setup to Pandora, via `LArPandora`:~~ From the discussions here, sounds like utilising Intel TBB would be much easier for us, whilst also being the recommended way. Requires setting up an optional TBB dependency, moving current code to use it.

After this, the longer term steps are those outlined previously, not necessarily in any order:

- Look at allowing the individual algorithms to be parallelised, rather than the higher level workers.
- Identify candidates for algorithms that can run in the background, where the output is only needed much later.
- Continued performance verification to understand bottlenecks and potential places to improve performance or parallelise the workflow.

Conclusion

In conclusion,

- Despite not currently using threads, Pandora is in a good position to start using them effectively: There has been considerations for threading in Pandora for many years, and both the technical implementation and the workflow of Pandora lends itself to threading.
- There is many potential options where to exploit threading in Pandora, ranging from simple changes to changes that will require more thought.
- As a first look, the cosmic ray workers Pandora utilises have been parallelised and tested at ProtoDUNE-SP.
- We see a decent performance increase, even with the caveats around the style of parallelisation.
- We have a roadmap for further improvements to Pandora.

Threading in Pandora

Thoughts & First Look

Ryan Cross - LArSoft Threading Workshop
2023/03/03

WARWICK
THE UNIVERSITY OF WARWICK

Backup Slides

reco_one_part_one Stage

Module	Min	Avg	Max	Median	RMS
Full event	123.29	201.247	362.061	199.083	49.4361
DataPrepModule	4.27057	4.51445	6.30109	4.44527	0.333081
WireCellToolkit	43.367	47.1386	49.7882	47.5742	1.35936
GausHitFinder	0.679067	1.13265	1.76745	1.15863	0.231892
SpacePointSolver	4.25527	11.5778	66.6611	9.60481	9.6333
DisambigFromSpcePnts	0.580875	1.31627	2.61369	1.29734	0.415445
StandardPandora	**15.1198**	**35.9972**	**98.6899**	**32.0013**	**17.5274**
LArPandoraTrack	1.10726	2.95595	5.97499	2.83599	0.949648
LArPandoraShower	1.61451	5.13073	12.1788	5.13466	2.20427
Calorimetry	0.63022	1.39994	2.70598	1.37807	0.447283
Calorimetry	0.557769	1.38169	2.70228	1.33677	0.451757
ShowerCalorimetry	1.67604	4.98912	11.7536	4.94189	2.18328
ShowerCalorimetry	1.59542	4.78268	11.6005	4.65735	2.14259

reco_one_part_two Stage

Module	Min	Avg	Max	Median	RMS
EmTrackMichelId	24.0752	38.9504	59.5697	39.1083	7.65593
LArPandoraTrackCreation	4.63419	11.9689	25.3461	11.7338	4.58386
Calorimetry (SCE)	1.94079	4.93515	10.4716	4.80801	1.97635
Calorimetry	1.91416	4.74727	10.0857	4.68517	1.86267
LArPandoraShowerCreation	1.82891	4.81261	9.72359	4.97146	1.76673
ShowerCalorimetry (SCE)	1.99548	5.00194	9.8497	5.10823	1.89068
ShowerCalorimetry	1.93357	4.66104	8.77288	4.71526	1.69058