SPNG Tensor Data Model and Classes

Brett Viren

September 5, 2025

Topics

- SPNG Tensor Data Model
- Helper classes
- Data flow graph node classes

A starting reminder:

- `ITorchTensor` has **metadata** and (torch) **tensor** parts.
- `ITorchTensorSet` has **metadata** and a **vector** of `ITorchTensor`.

The SPNG tensor data model addresses these highlighted aspects.

Relevant issues: #7, #8, #14 (tangentially), **#19**, #21.

SPNG Tensor Data Model

Documented at

https://github.com/WireCell/wire-cell-toolkit/blob/spng/spng/docs/datamodel.org

Overview

- Goals: specify/standardize content of `ITorchTensorSet` / `ITorchTensor` in SPNG
- Based on the "WCT Tensor Data Model" for `ITensorSet` / `ITensor`.
  https://github.com/WireCell/wire-cell-toolkit/blob/master/aux/docs/tensor-data-model.org
- Covers **metadata** and **tensor** part with care for Torch idioms (eg batching).
- Data model is split into **low-level** and **high-level** sub-models.

Caveat

"Non-TDM" tensors are still allowed but I hope we move SPNG to be largely "TDM compliant".

- Benefits: simplify our thinking and our coding.

Low-level SPNG tensor data model

`ITorchTensorSet` low-level constraints - **almost none**

Except: a "parent" tensor must precede any of its "children" in the **vector**.

`ITorchTensor` low-level **metadata** attributes

Required:

- datatype: how to **interpret** a tensor.
- datapath: how to **uniquely identify** the tensor in a set.

Optional:

- parent: the **datapath** of another tensor that logically "contains" this one.
- batches: integer giving size of batch dimension size, omitted for no batching.

There are no low-level specification for `ITorchTensor` **tensor** part.

High-level SPNG tensor data model

Each **datatype** implies additional specification. So far only `IFrame`-related are defined.

"frame": non-constituent parts of an `IFrame`

No tensor part. Metadata: **ident**, **time**, **period**.

"traces": dense set of channel vs time waveform samples

Tensor part holds the samples, can be batched. Metadata: **parent** (a "frame" datatype), **tag**, **tbin0**.

"chids": channel IDs of traces

Tensor part holds channel IDs, can be batched. Metadata: **parent**, **tag**.

"summaries": summary vector of traces

Tensor part: holds summaries, same shape as chids, can be batched. Metadata: **parent**, **tag**.

"chmasks": channel masks maps

Tnsor part: 2D with columns: (batch index, chid, tickbeg, tickend). Metadata: **parent**, **label**.

SPNG classes to support SPNG TDM

Helpers

`TensorIndex` parentage tree and direct lookup by **datapath.**

`TensorSelector` configurable rule based selection of tensors.

`TensorRenaming` configurable rule based renaming of **datapath**.

`TorchContext` configurable `device()` and semaphore guarded context manager.

`ContextBase` glue between `TorchContext` and a DFP graph node class.

Data flow nodes

`FunctionNode` apply standard, configurable selecting/transforming/renaming.

`TorchFunctionNode` `TorchContext` plus no-auto-grad guarded method.

`FanoutNode` a **general purpose** fan-out for `ITorchTensorSet`.

`FaninNode` a **general purpose** fan-in for `ITorchTensorSet`.

The `TensorIndex`

- Fill with `ITorchTensorSet`, caries `ident` and `metadata` along for the ride.
- Organizes the tensors into a n-ary tree according to **parent** attribute.
- Used by selection and renaming to assure consistency is kept.
- Fast lookup by **datapath** attributes.

We may later use the tree structure to help collect tensors into explicit structure.

- *eg* we may consider an `IFrame`-like assembly to help deal with the TDM parts.

The `TensorSelector`

Standardized, configurable selection rules.

- Array of objects with `accept` and/or `reject` attributes.
- Specified as `std::regex`.

Example usage: follow an N-way fanout with N selectors to convert a per-APA frame and into many per-plane frames.

The `TensorRenamer`

Standardized, configurable renaming rules.

- Array of objects with `match` regex and `replace` pattern attributes.
- The `match` is specified as `std::regex`, the `replace`
  - can use regex "group variables" (`$1`, `$2`, etc)

Example usage: precede an N-way fanin with N renamers to assure unique **datapaths**.

A comment about using `std::regex`.

For me, regex is hard, but powerful. There is a command line helper to get your regex correct:

Selecting

```
$ ./build/util/check-regex "/path/to/dummy" '.*/dummy$'
$ echo $?
0
```

Renaming

```
$ ./build/util/check-regex "/path/to/dummy" '(.*)/dummy' '$1/smarty'
/path/to/smarty

$ ./build/util/check-regex "/path/to/frame/dummy" '^.*/(.*)/(.*)$' '/swapped/$2/$1'
/swapped/dummy/frame
```

The TorchContext and TorchSemaphore and ContextBase

Configurable Torch device and semaphore context manager.

device

Use `TorchContext::device()` method to return user-configurable Torch device.

semaphore

Use `TorchContext::{enter,exit}()` to manually engage semaphore guard.
Or, use code scope guard:

```
{
  TorchSemaphore guard(my_torch_context);
  // ...do Torch operations safely here...
} // the guard is released
```

context base

`ContextBase` turns `TorchContext` into a base class. Used by `TorchFunctionNode`.

SPNG TDM data flow graph node classes

Most helper classes are invisible and instead we write flow graph nodes.

`FunctionNode` a **base class** apply standard, configurable selecting/renaming.

`TorchFunctionNode` a **base class** for function nodes using Torch operations.

`FanoutNode` a **general purpose** fan-out for `ITorchTensorSet`.

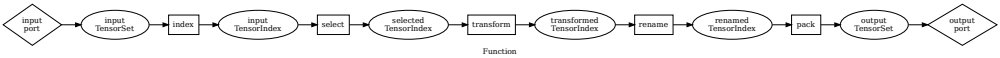
`FaninNode` a **general purpose** fan-in for `ITorchTensorSet`.

Common features:

- Standard log messages giving data state.
  - Quelled with `quiet = true` configuration.
- Usable as-is and extensible by inheritance.

`FunctionNode` internals

index → select → transform → rename → pack



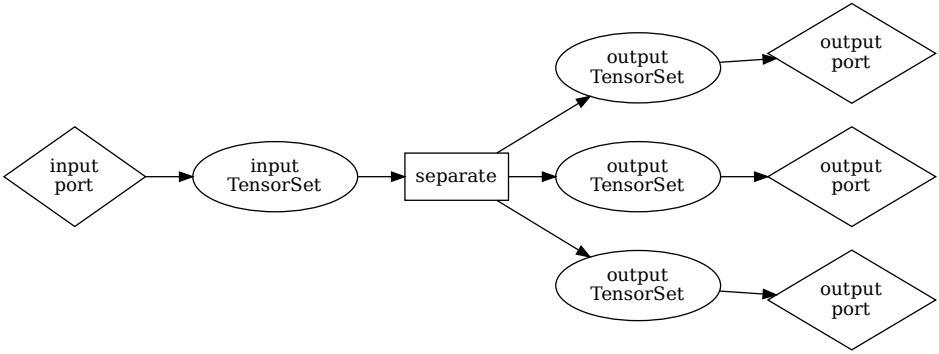
Should inherit and override `transform_function()`.

- May override any other methods, but usually should not.
- The method filters a `TorchIndex` (not a tensor set).

`TorchFunctionNode` looks identical.

- It assures tensors are on the configured Torch device and then,
- It calls `transform_function()` in a guarded torch context.

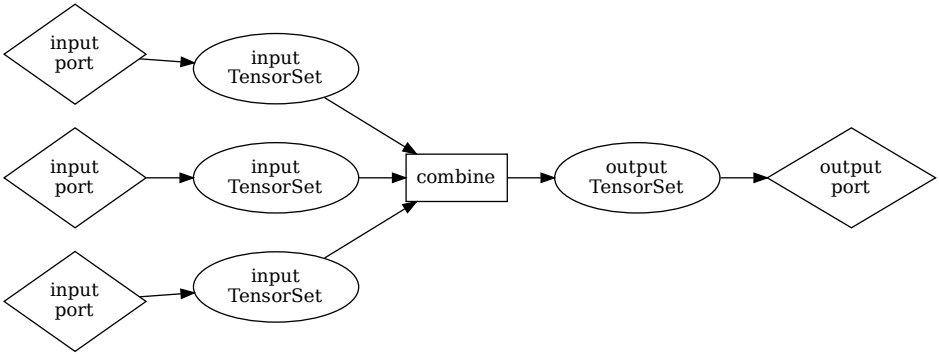
Fanout internals



Fanout

May, but usually should not, inherit and override `separate_tensors()`.

`Fanin` internals



Fanin

May, but usually should not, inherit and override `combine_tensors()`.

Inheriting from the SPNG TDM DFP graph node classes

`TensorIndex transform_tensors(TensorIndex ti) const`

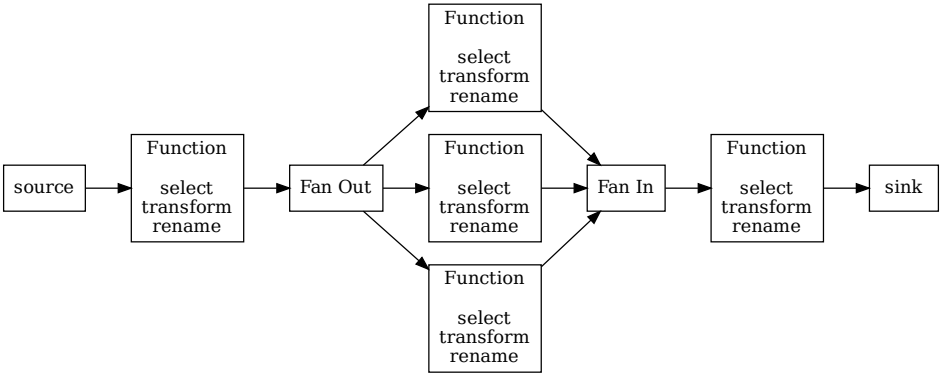
- The base `FunctionNode` is useful as-is but both may be used as base classes.
  - `TorchFunctionNode` can run as-is, but adds unnecessary overhead.
- Both are intended to serve as base classes.
  - Use `FunctionNode` if no Torch operations, else `TorchFunctionNode`.
- The `TensorIndex` is after selection, before renaming.

Fans

May subclass and provide “separate” or “combine” but try to avoid that.

- Instead, combine fans with functions (see next example).
- Note, fans work on `ITorchTensorSet` not the more convenient `TensorIndex`

High level operations with simple fans, full featured functions



General, primitive subgraph with general fans, specific functions.

Guidance: apply transformation in functions, leave the fans generic.

To do plan

First, does this all seem good? Any problems/improvements? Good to go? If so, nominal plan is:

Me: retrofit a few existing node classes.

- Start simple, shake out any remaining design problems.
- Then, `FrameToTorchSetFanout` and `TorchTensorSetToFrameFanin`
  - I still need to carefully study their current implementations.
  - Can they be usefully factored into separate function+fan nodes?
  - At least we can assure the input/output is a TDM-compliant tensor set.

All: embrace the TDM

- Help retrofit the remaining existing nodes.
- Use TDM for any new nodes nodes.

Related

- Thread safe tensors via stream guards (issue #8)
- Develop TDM-compliant file I/O nodes (#14).