An Introduction to PHASM

SRGS 2022

Nathan Brei, David Lawrence

Jefferson Lab

June 27, 2022

Replacing existing code with neural networks

A neural network is a numerical method that can approximate any computable function. Sometimes we might prefer a neural net approximation over an algorithm that gives an "exact solution".

Potential advantages

- Runs faster/with less memory (transform time/space complexity)
- Runs on a GPU or FPGA (transforms the inherent parallelism)
- Runs backwards (inverse problems)

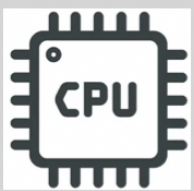
Challenges

- Matching the *order of accuracy* of the original
- Bounding the accuracy
- Predicting the accuracy *(uncertainty quantification)*
- Avoiding the *Curse of Dimensionality*

The heterogeneous hardware situation

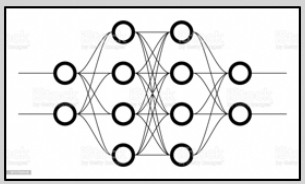
Not so long ago

```
template <typename R,
typename D> class
fixer { public: R
fix(D x) { return
f(x); } private:
virtual R f(D) = 0;
}; class fact :
public fixer<long,
long> { virtual long
f(long x) { if (x ==
0) { return 1; }
return x * fix(x-1);
} }; long result =
fact().fix(5);
```



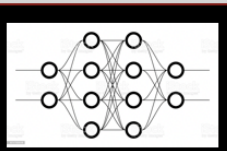
Now

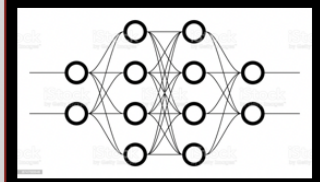
```
template <typename R,
typename D> class
fixer { public: R
fix(D x) { return
f(x); } private:
virtual R f(D) = 0;
}; class fact :
public fixer<long,
long> { virtual long
f(long x) { if (x ==
0) { return 1; }
return x * fix(x-1);
} }; long result =
fact().fix(5);
```



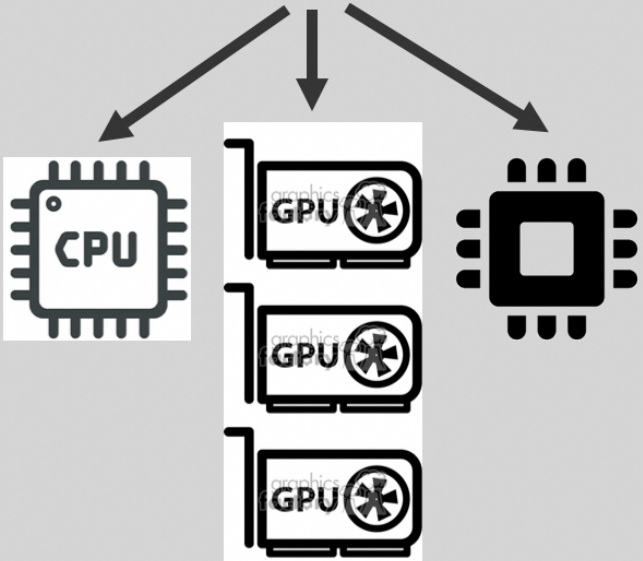
Soon

```
template <typename R,
typename D> class fixer {
public: R fix(D x) {
return f(x); } private:
virtual R f(D) = 0; };
```





```
class fact : public
fixer<long, long> {
virtual long f(long
x) { if (x == 0) {
return 1; } return x
* fix(x-1); } }; long
result =
fact().fix(5);
```



What is PHASM?

PHASM *(Parallel Hardware and Surrogate Models)* is an experimental toolkit focused on getting old, crufty, legacy code to run on shiny, new, highly parallel hardware, by replacing the most intensive parts of the code with neural net surrogate models. There are three main pieces to this:

- Figuring out how to do a *performance analysis* to figure out if a surrogate model even makes sense for a given function.
- Automatically figuring out what all of the inputs and outputs are for a given function. (You would think this would be obvious, but it's not)
- Create an interface for integrating surrogate models into existing C++ codebases as cleanly and abstractly as possible. (So cleanly that eventually a machine could do it for us!)

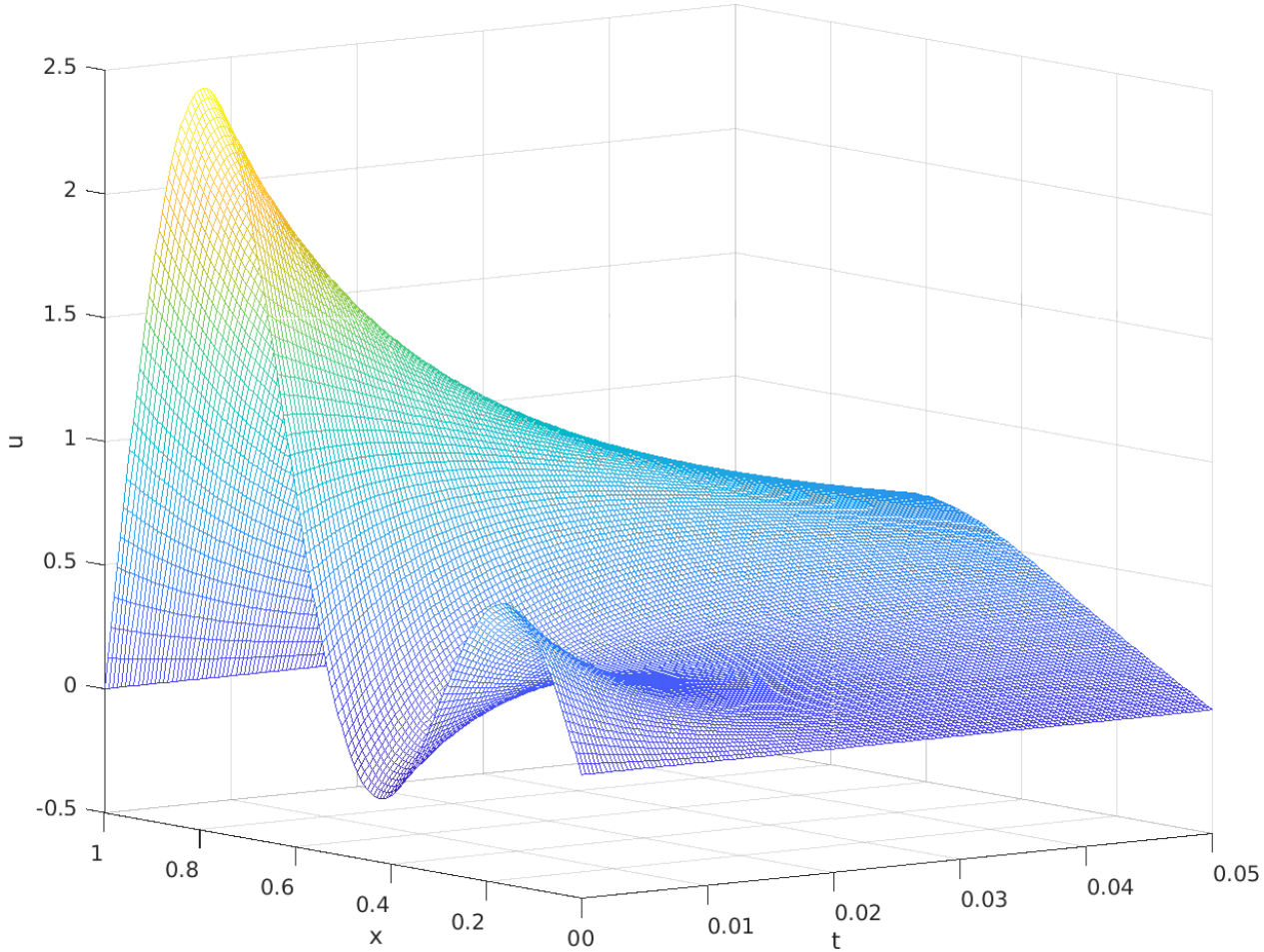
PHASM ML models

Right now, PHASM is missing an important piece: neural net models that do one or more of the following:

- Are constrained to produce physically valid solutions
- Don't need lots of training data
- Can be trained to a bounded level of accuracy
- Know their own uncertainty
- Can accept inputs of different sizes

We are particularly interested in ordinary and partial differential equation solvers.

Heat equation PDE

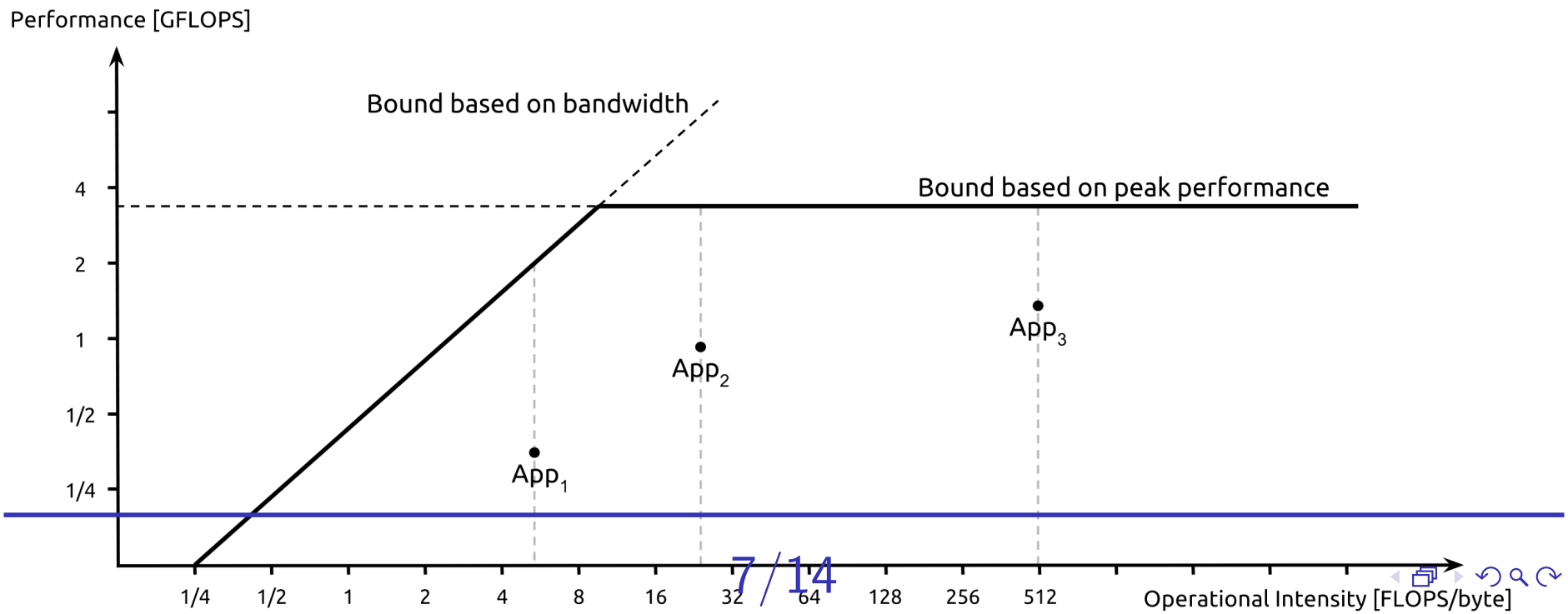


Goal

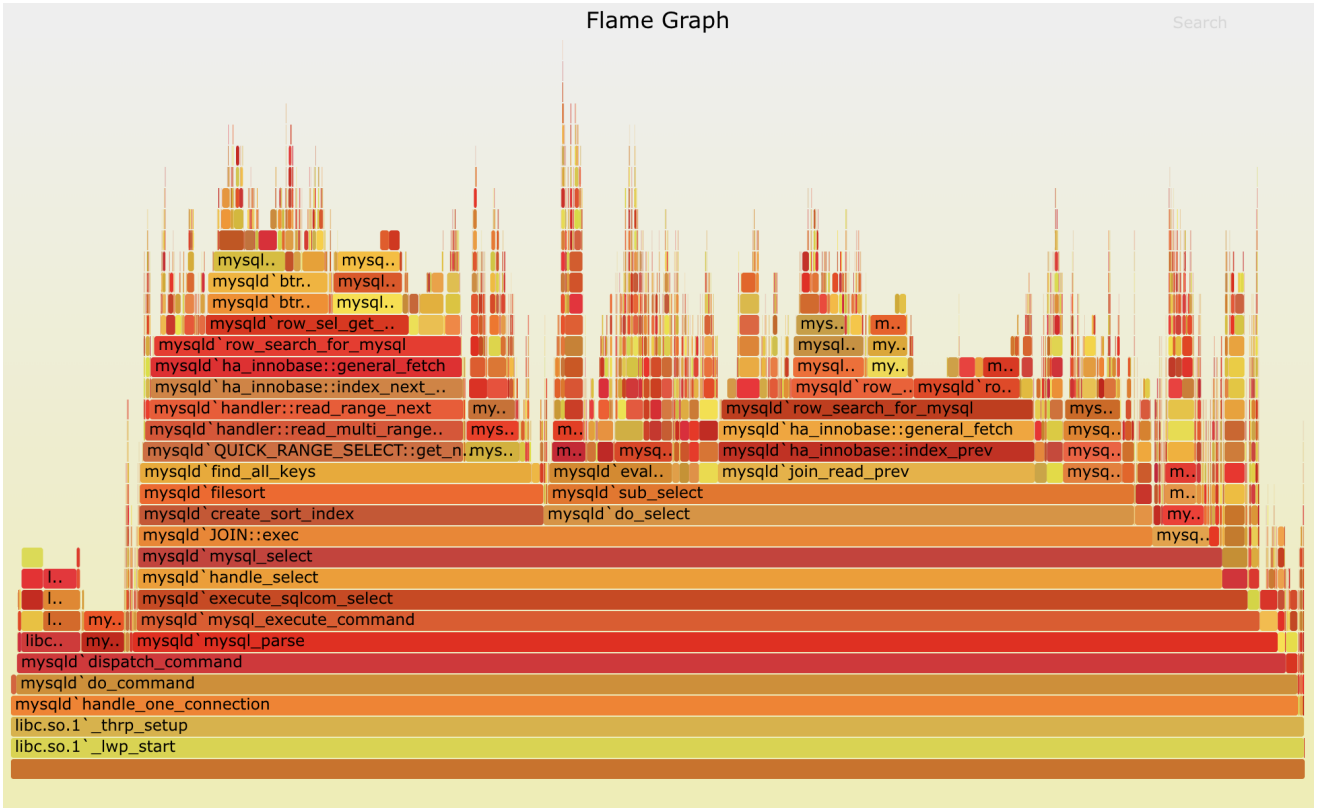
Take the guesswork out of determining when (and later where) it makes sense to deploy a surrogate model.

Deliverable

A playbook that describes a decision procedure using performance metrics obtained from off-the-shelf profiling tools.



Performance analysis



Goal

Automatically identify and extract the necessary model variables when creating a surrogate for an arbitrary target function.

Deliverable

A tool that traces all memory accesses and reports which variables were read and written from within the target function.

Goal

Provide an API to insert a surrogate model into a legacy C++ codebase and interact with it in a clean, abstract, and pragmatic way.

Deliverable

A tool that traces all memory accesses and reports which variables were read and written from within the target function.

Surrogate API example

```
double f(double x, double y, double z) {
    return 3*x*x + 2*y + z;
}

// We create a surrogate model for f like so:

phasm::Surrogate f_surrogate = phasm::SurrogateBuilder()
        .set_model(std::make_shared<phasm::FeedForwardModel>())
        .local_primitive<double>("x", phasm::IN)
        .local_primitive<double>("y", phasm::IN)
        .local_primitive<double>("z", phasm::IN)
        .local_primitive<double>("returns", phasm::OUT)
        .finish();
```

Surrogate API example

```
double f_wrapper(double x, double y, double z) {
    double result = 0.0;
    f_surrogate.bind_original_function([&](){ result = f(x,y,z); })
               .bind_all_callsite_vars(&x, &y, &z)
               .call();
    return result;
}
```

Research Roadmap

- Become familiar with PHASM and PyTorch by running and playing with the magnetic field map example. Create a model in PyTorch (preferably in JupyterLab), export to TorchScript, and run it using PHASM.
- Write a ODE solver that ‘swims‘ a particle through a magnetic field. First use a constant magnetic field, then use the magnetic field map surrogate model from before.
- Write a PDE solver for the 1D (and then 2D) diffusion equations using finite differences. Surrogate it using a convolutional neural net inside PHASM.
- Try to encode the equations for the particle swimming ODE directly in the magnetic field map neural net. Try to encode the equations for the diffusion PDE directly in the neural net itself.

This is research, not development! We constantly communicate what is working for us and what isn't, and adjust the plan accordingly.