THE UNIVERSITY of EDINBURGH

epcc

Using the Cerebras CS-2 for scientific computing via PETSc

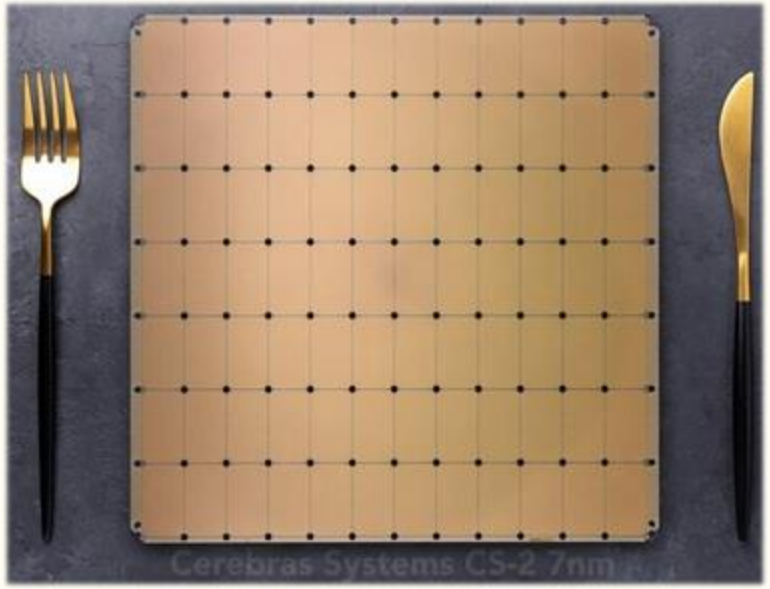
Justs Zariņš, Joseph Lee, Michèle Weiland

j.zarins@epcc.ed.ac.uk

Overview of Cerebras CS-2

Wafer Scale Engine (WSE-2)

- Whole wafer chip (7nm technology)
- 850,000 cores (individually programmable)
- 40 GB on-chip memory
- 20 PB/s aggregate memory bandwidth
- Supports up to 32-bit floats and ints in HW

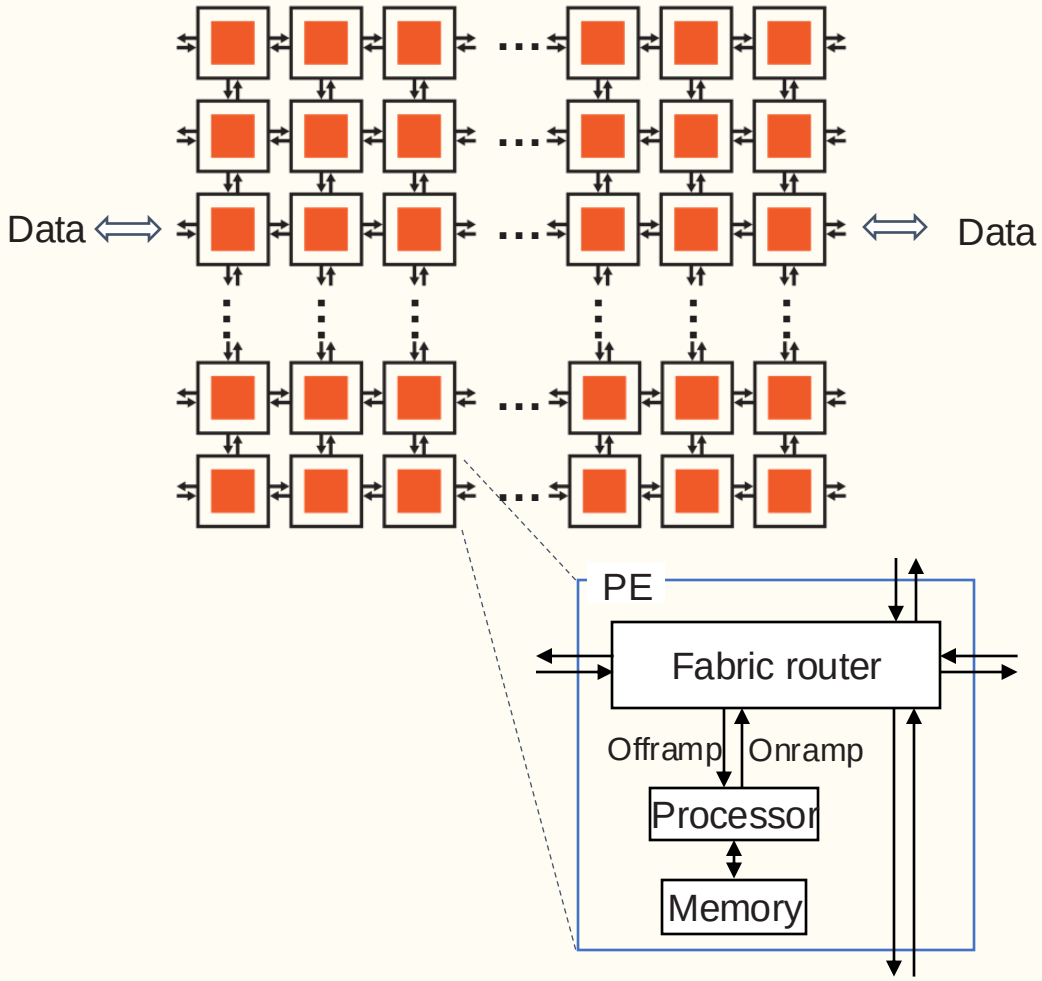


Cerebras CS-2 system

- Custom power and cooling solution to support the WSE
- Fits into standard racks (15RU)
- 23 kW of power
- Multiple can be networked together

WSE architecture

- 48 kB SRAM per processing element (PE)
- PEs arranged in 2D grid
- 32-bit messages (wavelets) can be communicated with **neighbours** in a single cycle
- Programmable router in each PE



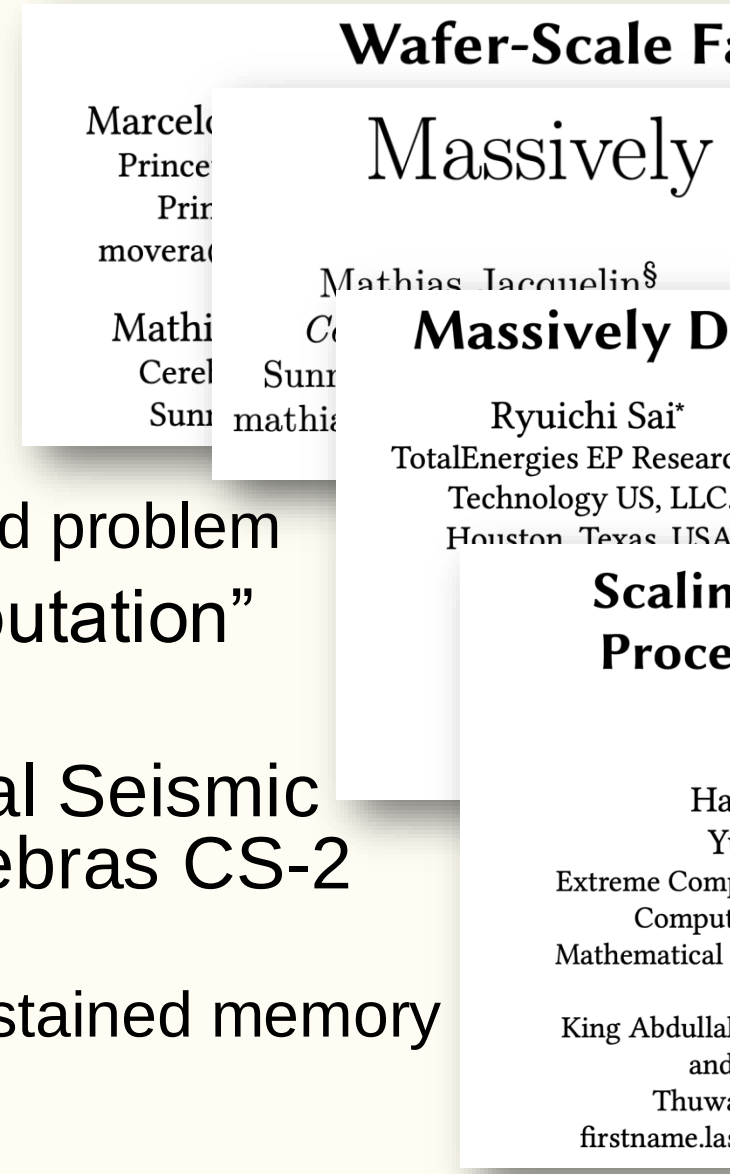
What types of application suit the WSE?

- Appliaction is scaling poorly across multiple nodes (e.g. FFTs, particle simulation)
  - High bandwidth and low-latency fabric
  - 850k cores on an individual chip instead of distributed across many nodes

- Application is constrained by data access
  - 40 GB of SRAM uniformly distributed across the wafer that can be accessed in 1 cycle
  - Up to 1.2 Tb/s bandwidth onto the chip

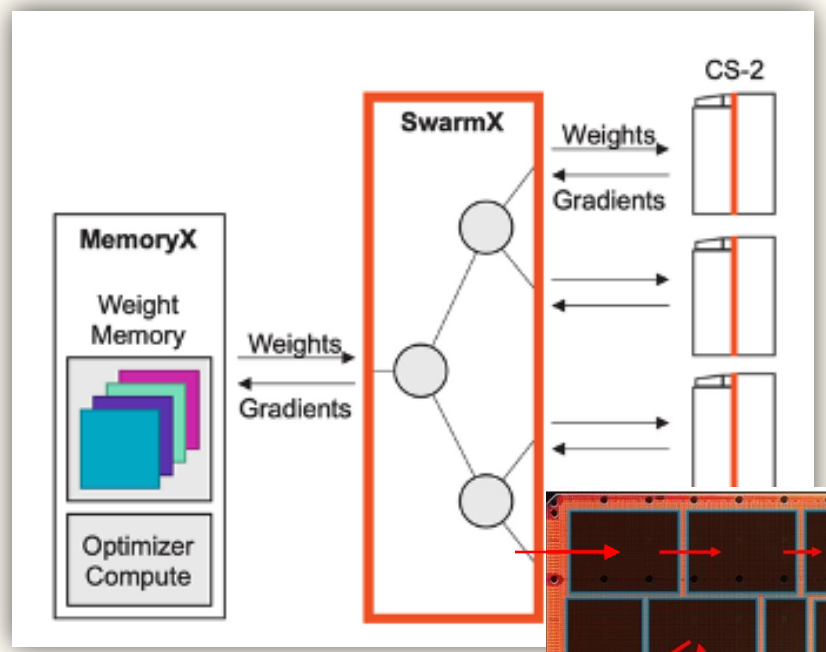
Success stories

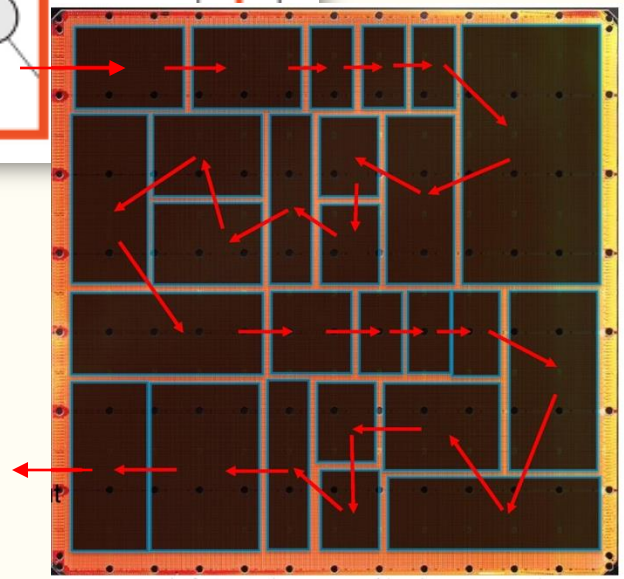
- "Wafer-Scale Fast Fourier Transforms"
  - fastest time for a usefully sized benchmark
- "Massively scalable stencil algorithm"
  - turns a memory bound problem into a compute bound problem
- "Massively Distributed Finite-Volume Flux Computation"
  - two orders of magnitude speedup over GPU
- "Scaling the "Memory Wall" for Multi-Dimensional Seismic Processing with Algebraic Compression on Cerebras CS-2 Systems"
  - Gordon Bell finalist on 48 CS-2 systems, 93 PB/s sustained memory bandwidth



Programming the CS-2 for AI

- High-level PyTorch interface
- Cerebras software automatically creates layer kernels and places them on the wafer
- Model size and training speed can be scaled independently in Cluster Mode



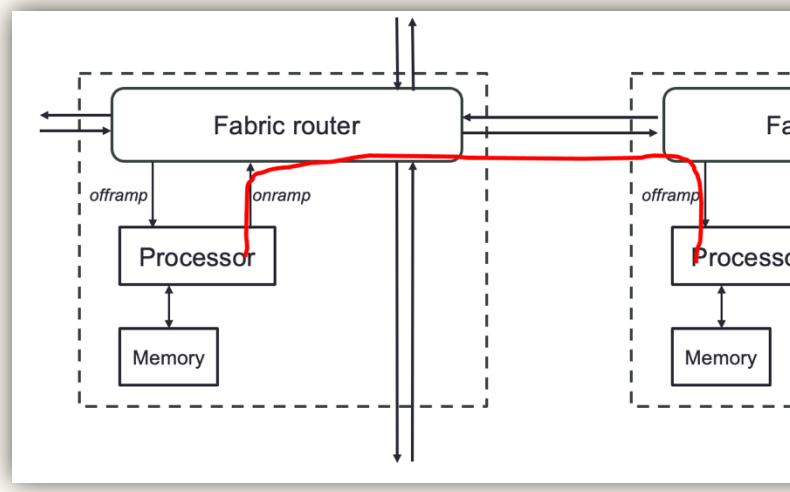


Programming the CS-2 for HPC

- HPC SDK available, including an accurate simulator for development
- **Dataflow** programming – tasks are activated by arrival of data packets
- Write kernels for PEs using CSL
- C/Zig-like syntax
- Standard: types, functions, control structures …
- Specific: builtins, modules, tasks, data structure descriptors …

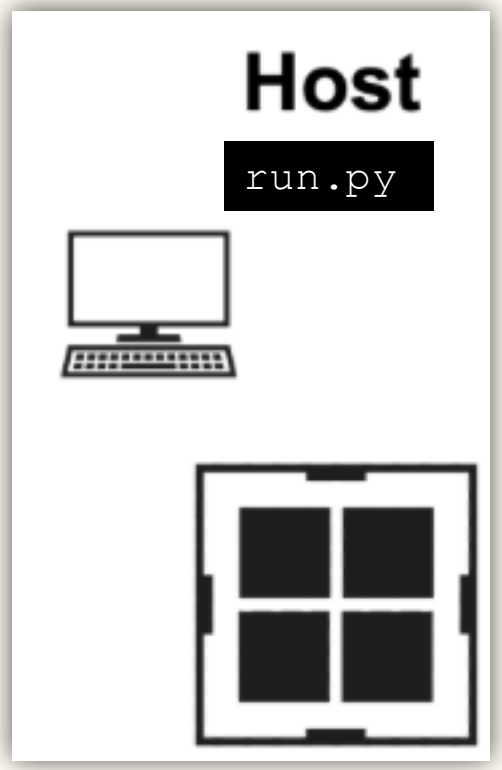
Programming the CS-2 for HPC

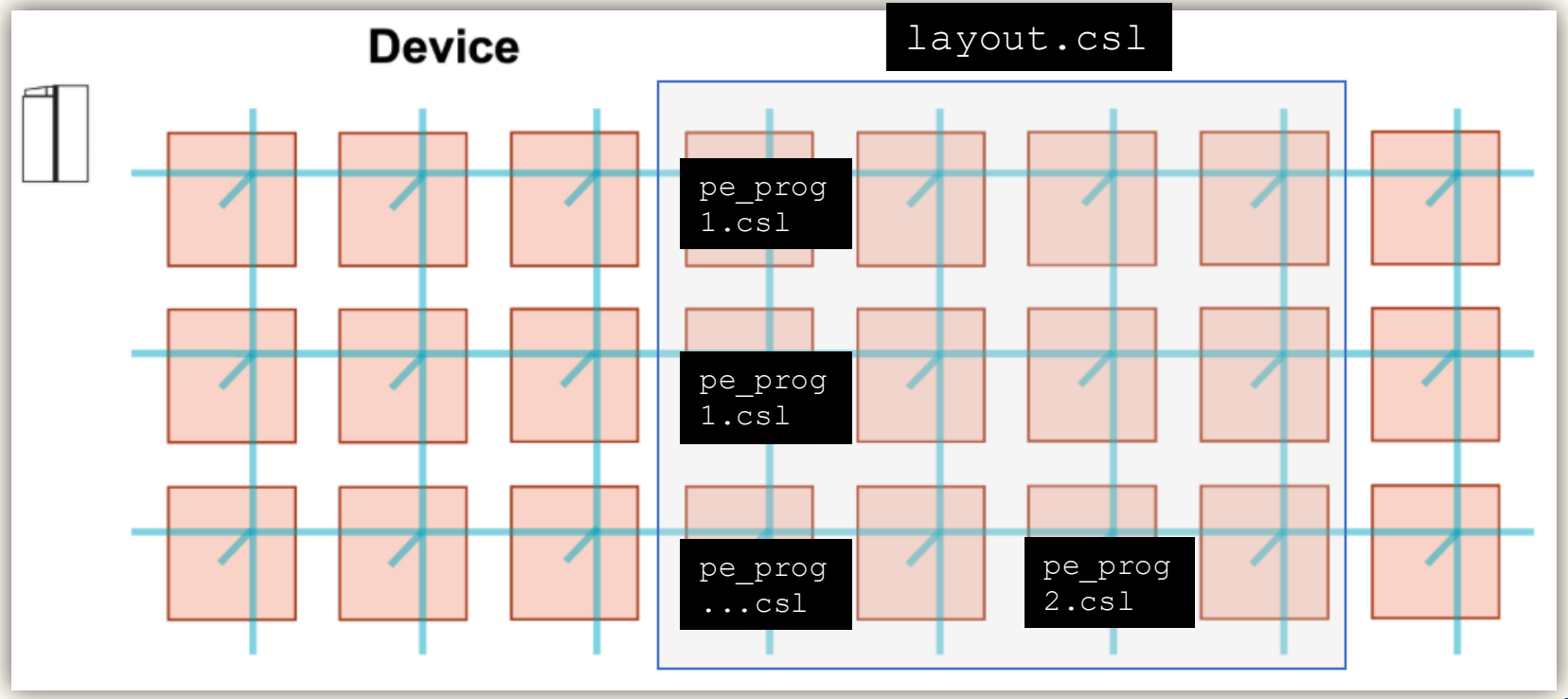
- Routers have access to ~24 virtual communication channels (colours) for passing messages (wavelets)
  - need to make sure send/recv are matching
  - easy to run out of colours
- Useful libraries available in the SDK
  - memcpy – “paste” data onto device
  - collectives – e.g. scatter and gather on device
- Debugger and Visualiser are available in the Simulator



Programming the CS-2 for HPC

- Python host code to drive the application
- Layout file to place kernels and set up routers





Using Cerebras CS-2

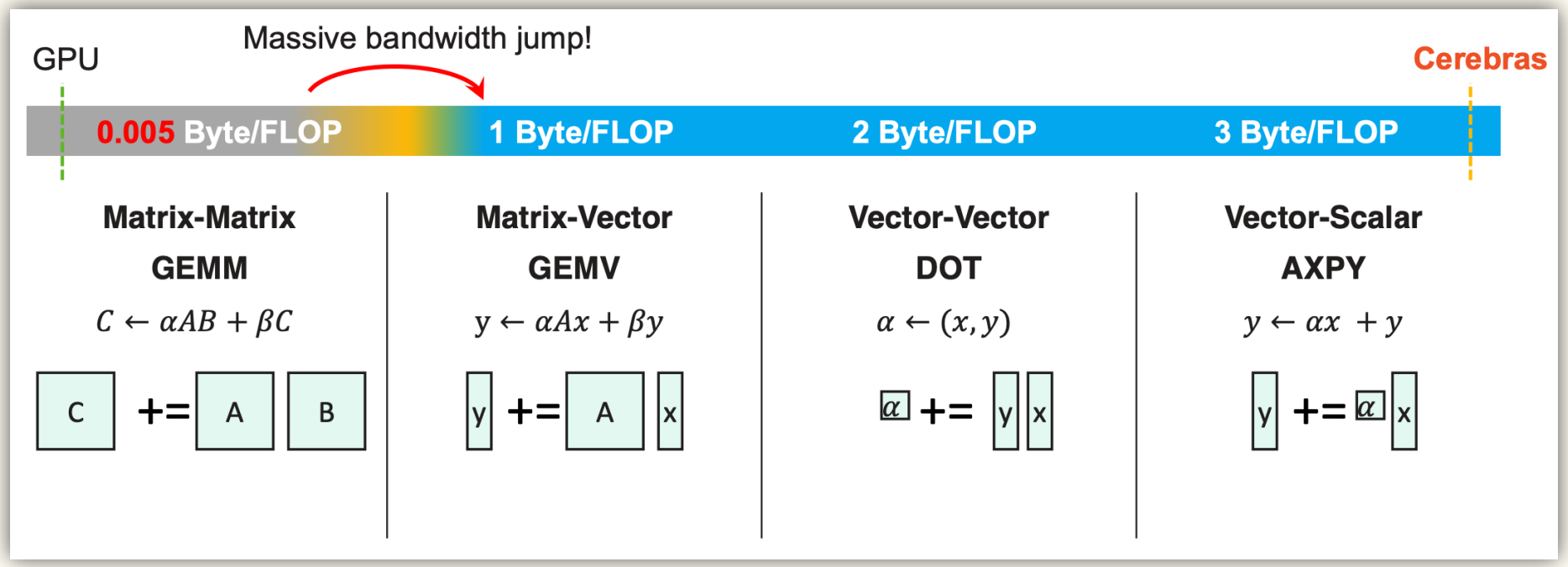
In practice …

- Quite tricky to program using the SDK!
- Dataflow paradigm fundamentally different from CPU/GPU
- Need to get multiple things right before any results are seen
- Hard to conceptualize the colour arrangements
- Challenging to debug deadlocks

- *Want to*: drive the WSE from a high level HPC application and reduce the barrier to using the CS-2
  - Explore PETSc as the interface

Using PETSc as the interface

- Linear Algebra is the basis for many HPC codes and matches the WSE architecture well



- Use petsc4py (because CS-2 host code is in python)

Using PETSc as the interface

- Implement a set of key kernels in CSL to build up higher level functions
  - Vector operations
  - Dense Matrix operations
  - Sparse Matrix operations
  - KSP solver, e.g. Conjugate Gradient
- Kernels are parameterised
  - fabric width and height
  - number of elements per PE

Using PETSc as the interface

```python
#from petsc4py import PETSc
from petsc4wse import PETSc

...
x = PETSc.Vec().createSeq(m)
y = PETSc.Vec().createSeq(m)
A = PETSc.Mat().createDense([M,N])
x.setValues(range(m), range(m), addv=None)
...
result = x.sum()
A.mult(x,y) # y=A*x
...
```

```python
from petsc4py import PETSc

# Subclass the Vec class and override its sum function
# to do the work using the WSE
class Vec(PETSc.Vec):
  def sum(self):
    x = self.array

    calc_wse_distribution()
    compile_for_wse()
    initialise_wafer()
    memcpy_h2d(x, ...)
    wse_remote_launch("vec_sum")
    memcpy_d2h(x_sum, ...)

    return x_sum

# Assign modified functions to the original module
PETSc.Vec = Vec
```

Programming challenges

- This interface works! But..
- Hard to write general/reusable code for CS-2
  - HPC API has room for improvement
  - Hardware resource constraints
  - Much easier to write specific code if you want best performance
- The WSE is very flexible in theory, but not everything is available in the API
  - route remapping
  - no default access to PE coordinates at runtime
- The SDK goes through big changes

Performance challenges

- (End-to-end performance study not yet undertaken)

```
 9     calc_wse_distribution()
10     compile_for_wse()
11     initialise_wafer()
```

- Choices to be made about kernel shape and data distribution
- Kernel compilation cost incurred at runtime
- WSE startup time is significant
- BUT high-level view of the interface has potential to amortise these costs
  - hide latencies with other work
  - combine kernels to optimise dataflow between them

Experience of using the CS-2 for HPC

- Rewriting algorithms for a dataflow model is not trivial
  - but performance gains can be significant
- Getting wavelet routing correct is error-prone
- Best suited for long running, single-purpose kernels
- Lots of repeated effort across WSE-using projects
- Published result reproducibility barriers
- Some confusion on multi-user cluster sharing with AI jobs
- Helpful support and active community

Future outlook

- CS-3 already here!
- Good software updates in the pipeline
  - C++ host code
  - More provide communication routines
  - Linear algebra routines
  - printf debugging
- Would benefit from community cooperation (e.g. libraries)
- HPC API will improve
  - Co-design required to get it right

Using the Cerebras CS-2 for scientific computing (via PETSc) : Summary

Positive 🎉	Challenges 😢
Performance potential	Programming model
Tools (e.g. simulator)	Big software changes
Continuous improvements	Unclear how best to integrate in large apps
Helpful support	Repeated/similar/irreproducible work
Active community	Documentation and training