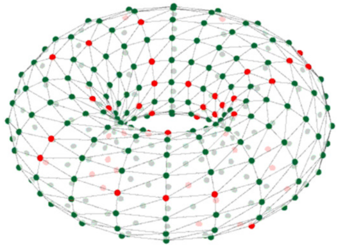
Writing an Application for SpiNNaker - Introduction



Simon Davidson, Alan Stokes, Andrew Rowley

SpiNNaker Workshop
September 2016

European Research Council

Human Brain Project

Contents

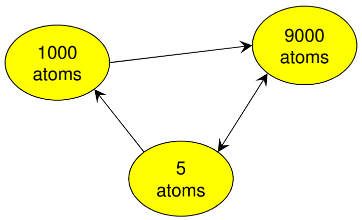
View of an application distributed across parallel processors

- Two main activities:
- Computation
- Communication

- Think of the problem as a graph:
- Vertex = computation node
- Edge = flow of information between nodes

- Node can hold a collections of objects of the same type, which we call ***atoms***

- e.g. Many spiking neurons in one population



Design Process for New Applications

- **The application designer** creates components (nodes and communication types)

- These components plug into our tool chain

- A **user** can then invoke the Graph Front End (GFE) to create and run their own networks on SpiNNaker

- Input is textual, like a PyNN script, in which the user instantiates the components created by the application designer

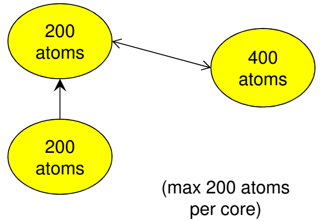
- Graph Front End is NOT a Graphical Interface - No GUI!

Example script: Conway's Game of Life

```
import spinnaker_graph_front_end as front_end
import sys

# set up the front end and ask for a machine with 48 chips
front_end.setup()

cell_1 = MyCell()
cell_2 = MyCell()
edge = MachineEdge(cell_1, cell_2)
front_end.add_machine_vertex_instance(cell_1)
front_end.add_machine_vertex_instance(cell_2)
front_end.add_machine_edge_instance(edge, "STATE")

# run the simulation for 5 seconds
front_end.run(5000)

# clean yp the machine for the next application.
front_end.stop()
```


Design Considerations I: Finite resources per core

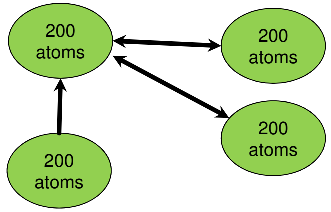
- The user's graph will be mapped to the cores of the SpiNNaker machine



- Each core has finite resource:
- Compute power
- Local memory
- Share of SDRAM capacity & bandwidth
- Communications bandwidth for packets

- Where each vertex represents many atoms we ***partition*** each one into smaller pieces, so that one piece fits on one core:

- **Application graph** maps to **Machine Graph**
- Edges also split to maintain correct connectivity



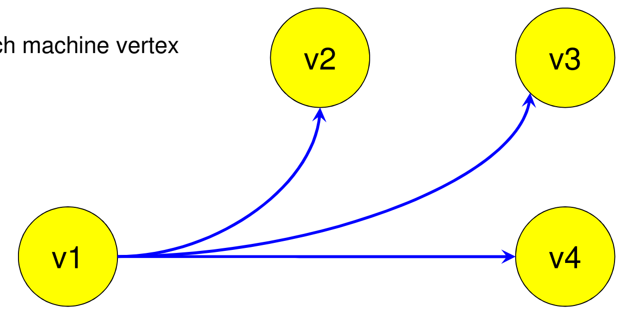
- Merging of vertices NOT currently supported!

Design Considerations II: Dataflow between Vertices

- Consider the pattern of messages flowing from each vertex:
- Case 1: Messages always go to the same set of targets
- Case 2: Messages go to different targets at different times

- Case 1: Homogeneous data flow
- e.g. spikes in neural simulation

- One **identifier** for each machine vertex



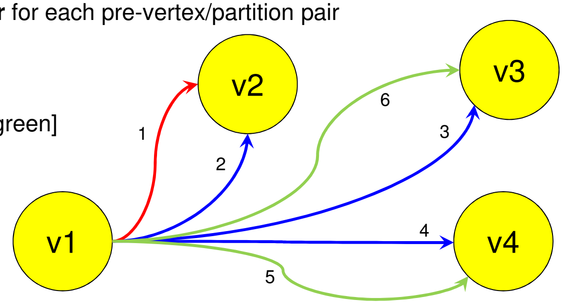
Design Considerations II: Dataflow between Vertices

- Case 2: Data send to different targets at different times:
- e.g. multi-layered perceptron, with forward and backward data flow
- Useful when there are different modes of operation

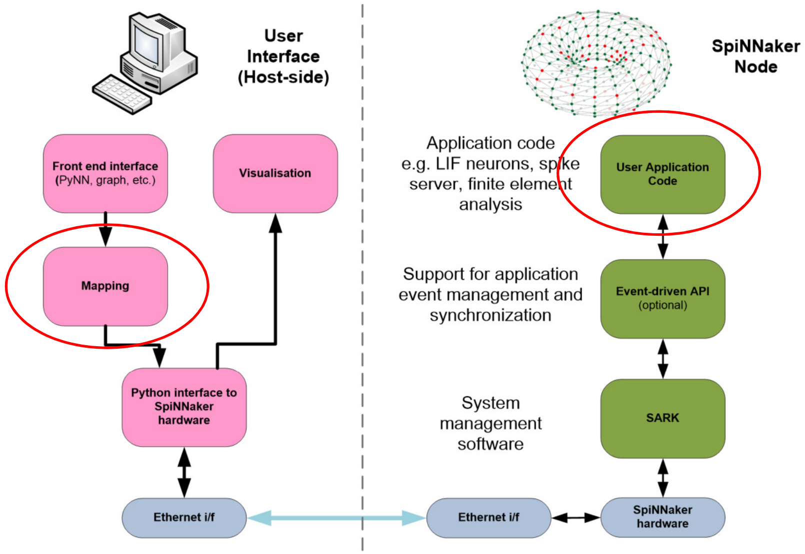
- Group edges so that those in same mode are together
- A grouping is called a **partition**

- Assign a separate **identifier** for each pre-vertex/partition pair

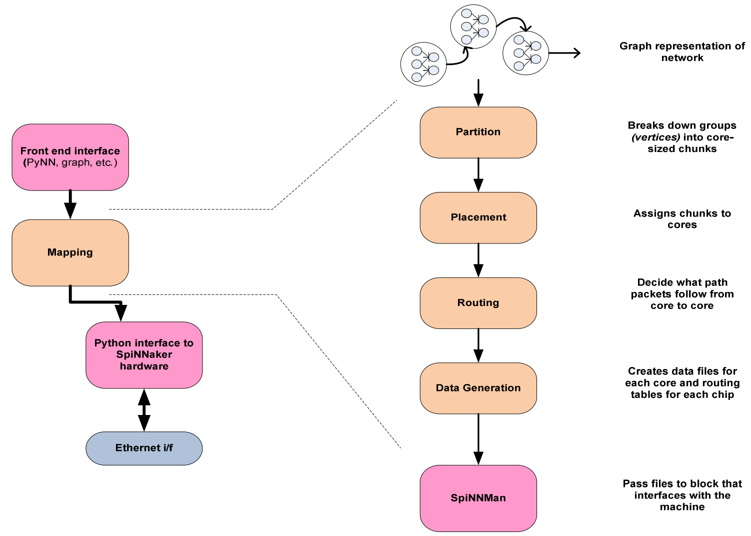
- Six edges [1, 2, .., 6]
- Three partitions [red, blue, green]



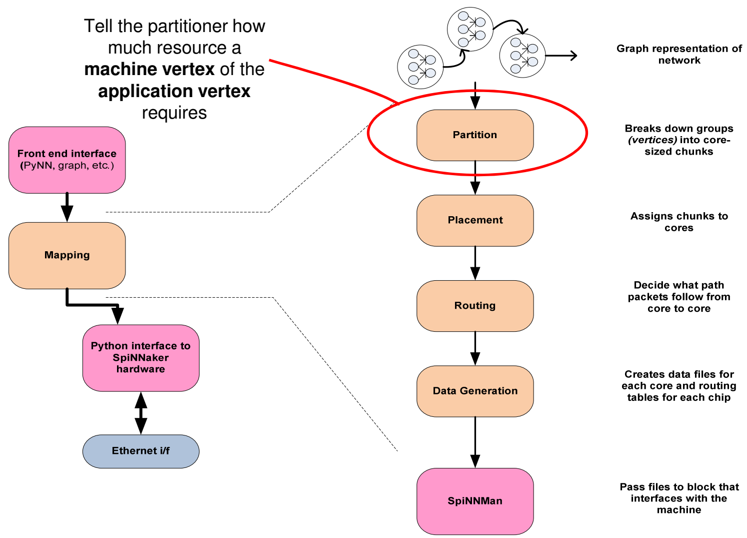
Software Stack



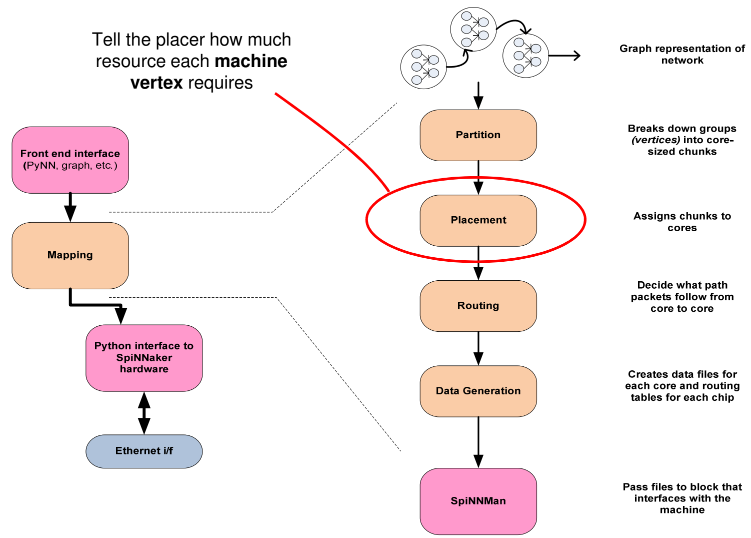
Where do you need to supply new information?



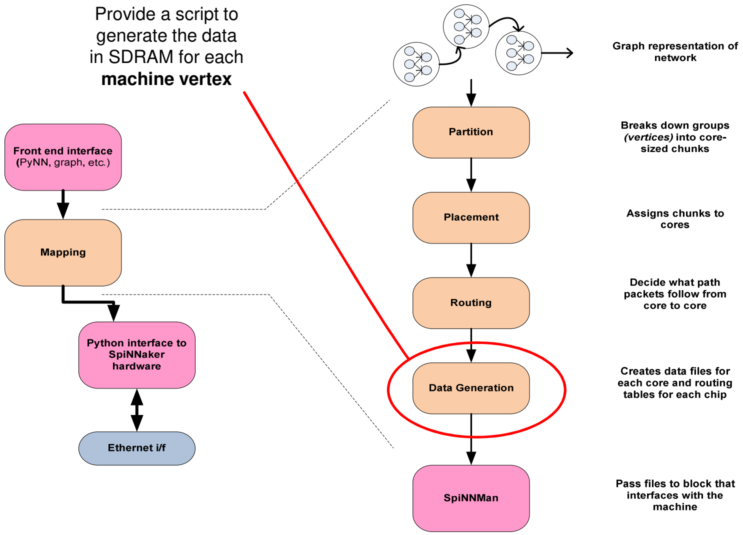
Where do you need to supply new information?



Where do you need to supply new information?



Where do you need to supply new information?



Data Spec. and Data Generation

- Each core running your application needs to generate its local data before it starts simulation
- We provide a simple ***virtual machine*** in which you can execute simple programs to generate this data
- This is the **Data Spec Executor (DSE)**
- The tools run code called the **Data Spec Generator (DSG)** that create a program (the specification or **spec**) for each core that is run by the DSG to generate its data

Summary

- It is useful to abstract any parallel application into the form of a graph with:
- Centres of computation (vertices)
- Connected by communication pathways (edges)

- **Application designer** must describe the computational elements and the communication types and plug those into our tools:
- Executables to run on SpiNNaker (typically written in C)
- Data specification, used to create each nodes data
- Describe resource requirements to allow tools to map networks to cores

- **User** can then specify application networks and run them using the Graph Front End.The tools handle :
- Mapping
- Routing table generation
- Data generation
- Loading
- Simulation
- Results gathering
- And other stuff ….