Australian National University

Vectorization in OpenCL

Vectorization

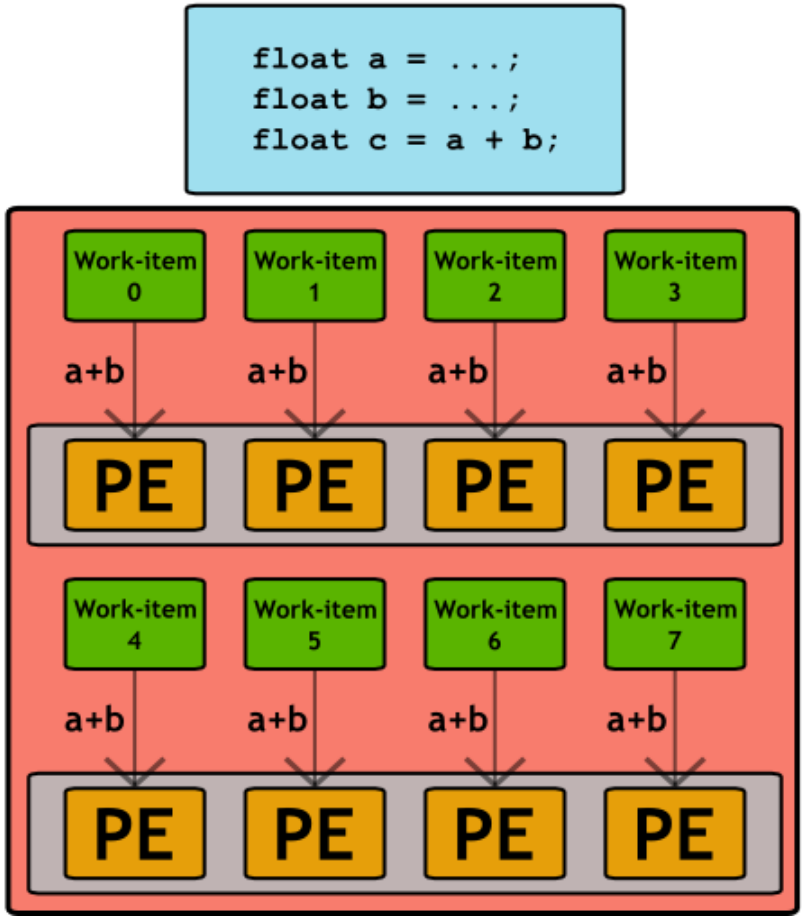
- OpenCL C provides a set of vector types:
  - `type2`, `type3`, `type4`, `type8` and `type16`
  - Where `type` is any primitive data type
- Than can be convenient for representing multi-component data:
  - Pixels in an image (RGBA)
  - Atoms or points (x, y, z, mass/type)
- There are also a set of built-in geometric functions for operating on these types (`dot`, `cross`, `distance`, `length`, `normalize`)

Vectorization

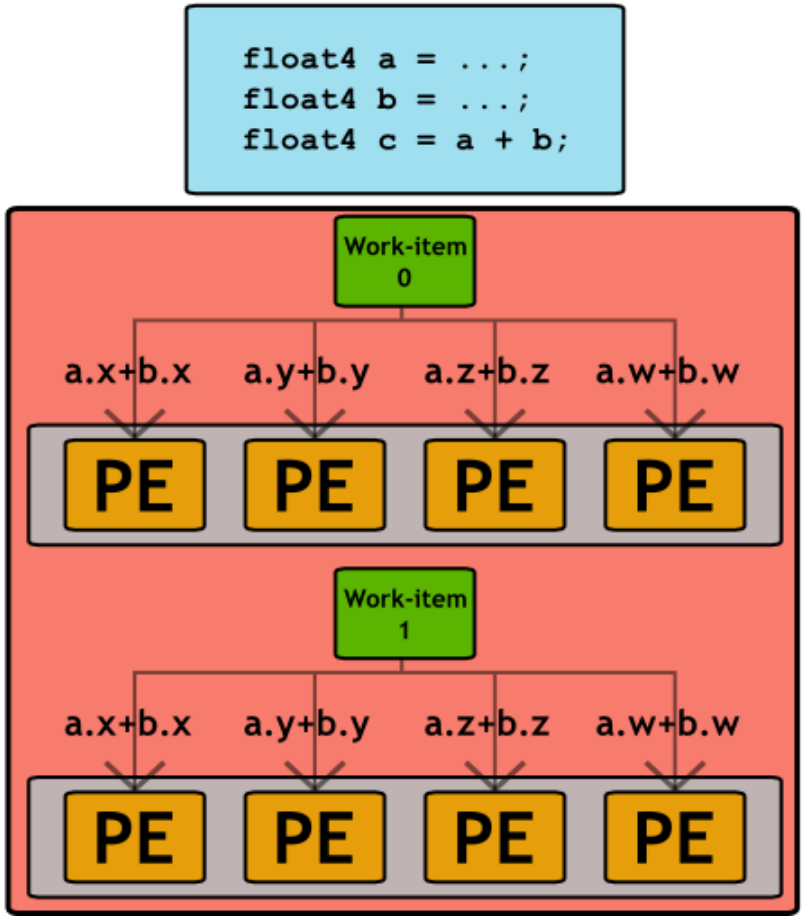
- In the past, several platforms ***required*** the use of these types in order to make use of their vector ALUs (e.g. AMD's pre-GCN architectures and Intel's initial CPU implementation)
- This isn't ideal: we are already exposing the data-parallelism in our code via OpenCL's `NDRange` construct – we shouldn't have to do it again!
- These days, most OpenCL implementations target SIMD execution units by packing work-items into SIMD lanes – so we get the benefits of these vector ALUs for free (Intel calls this '***implicit vectorisation***')

Vectorization

Implicit vectorization



Explicit vectorization



Vectorization

- You may come across some platforms that still require explicit vectorization
- As the architectures and compilers mature, we expect to see a continued shift towards simple, scalar work-items
- You can query an OpenCL device to determine whether it prefers scalar or vector data types, e.g:

```
clGetDeviceInfo(device,
  CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT,
  sizeof(cl_uint), &width, NULL);
```