Covariant programming : capturing the variation between SIMD and SIMT in a single code

The struct-of-array (SoA) portability problem:

- Scalar code: CPU needs struct memory accesses struct calculation
- SIMD vectorisation: CPU needs SoA memory accesses and SoA calculation
- SIMT coalesced reading: GPU needs SoA memory accesses struct calculation
- GPU data structures in memory and data structures in thread local calculations *differ*

Model	Memory	Thread
Scalar	Complex Spinor[4][3]	Complex Spinor[4][3]
SIMD	Complex Spinor[4][3][N]	Complex Spinor[4][3][N]
SIMT	Complex Spinor[4][3][N]	Complex Spinor[4][3]
Hybrid?	Complex Spinor[4][3][Nm][Nt]	Complex Spinor[4][3][Nt]

How to program portably?

- Use operator() to transform memory layout to per-thread layout.
- Two ways to access for read
- operator[] returns whole vector
  - operator() returns SIMD lane threadIdx.y in GPU code
  - operator() is a trivial identity map in CPU code
- Use coalescedWrite to insert thread data in lane threadIdx.y of memory layout.