**Motivation: Overheads of SPMD**

- SPMD languages (e.g., CUDA, OpenCL) are accessible and productive for programming data-parallel architectures
- GPUs reduce some overheads
  - Program counter, instruction fetch
  - But, redundancy across threads remains
  - Redundant loop bookkeeping, address calculations
  - Shared data
- Also, GPUs need to support **irregular control flow**
  - NVIDIA GPUs use a hardware reconvergence stack
  - AMD GPUs use implicit vector condition code registers

**How Vectors Deal with Redundancy**

1. Assume all variables are invariant
2. Mark source of variance (tid.x) as variant
3. Propagate variance forward

**Convergence Analysis**

- Dynamic hardware convergence preservation
- Combined convergence and variance analysis
- Simple structural convergence analysis
- Exit optimization

**Savings with Scalarization**

- 1.2x savings with scalarization (versus 2x)
- 1.0x converted warp parallel
- 1.1x converted warp dynamic
- 1.1x converted threads
- 1.1x microarchitecture action

**Results:**

- 31% Registers read (in paper)
- 30% Memory addresses and cache tag checks (in paper)
- 38% Memory data elements accessed (in paper)

**Conclusions**

- Maintain same SPMD programming model, but make “**scalarizing compiler**” generate vector-like code for better efficiency on GPUs
  - Use scalar register to attack redundancy
  - Use predicate registers to support **irregular control flow** more efficiently