Leveraging SIMD Architectures

"Vectorization for SIMD Architectures with Alignment Constraints"
-A. Eichenberger, P. Wu, & K O'Brien

"Efficient SIMD Code Generation for Runtime Alignment and Length Conversion"
- P. Wu, A. Eichenberger, & A. Wang

Presented by Peter Nelson and Dave Borel
February 27, 2007

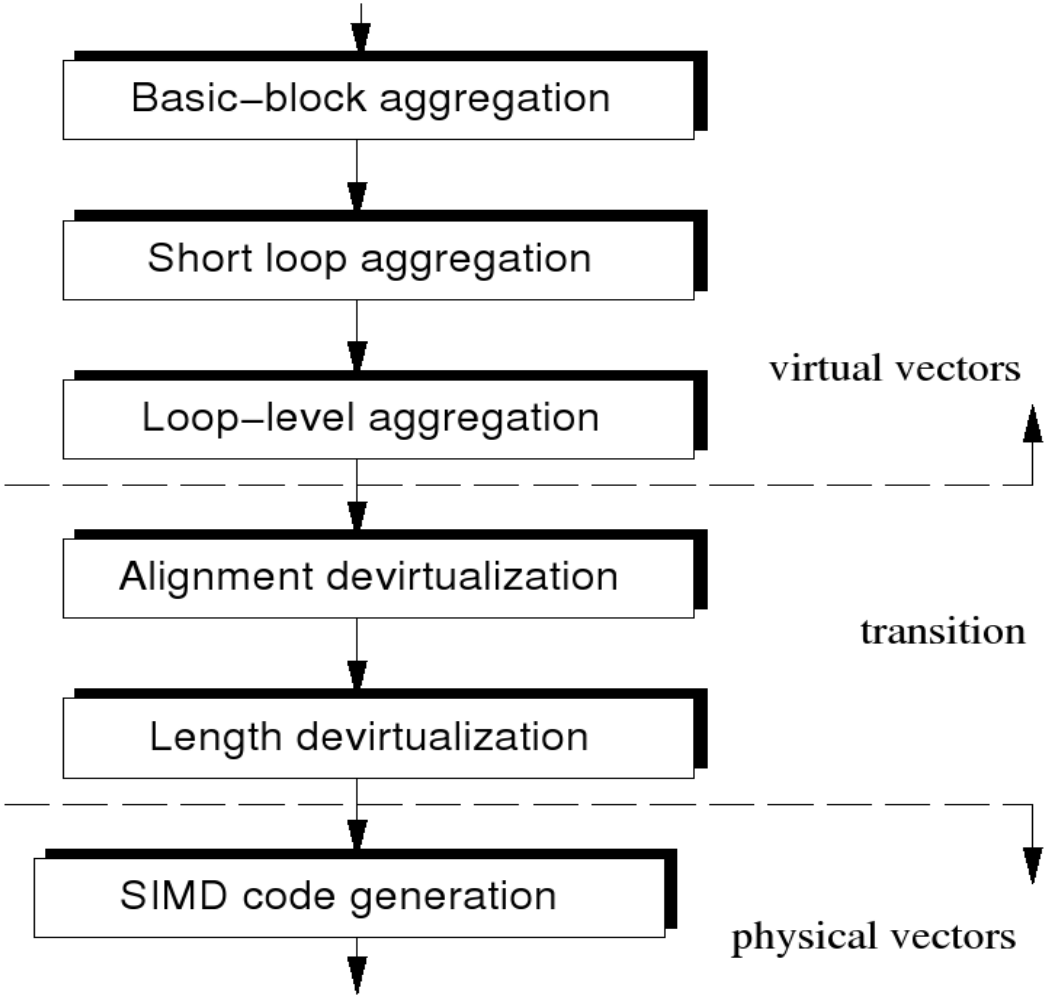
“Simdization”

- Vectors:
  - Data-level parallel sequences of scalars
- Implementations
  - Supercomputing
  - MMX, 3DNow!, SSEx, AltiVec
  - CELL
- SIMD:
  - Single Instruction, Multiple Data
- Things to consider
  - Data type, packing
  - Vector Length
  - Memory alignment

Classic Approach

- SIMD registers
  - *V* bytes each
  - *V*-byte aligned
  - *D* = sizeof(element)
  - Vector length *B* = *V* / *D*
  - Example: SSE – 16x8'b, 8x16'b, 4x32'b, 2x64'b
- Operations
  - parallel arithmetic (C = A .* B)
  - vector algebra (cross, dot, ...)
  - permute/shuffle/swizzle ({x,y,z,w} => {x,z,y,w}, ...)

CELL's Approach



“Virtual Vectors”/Streams

- Capture overall mathematical effect
  - Combine stride-one accesses
  - Support generic vector operations
  - Align sequence as a whole
  - Sign-extend

  ...defer SIMD instruction selection

Virtual Vector Aggregation

- Merge operations on contiguous data
- Pack “isomorphic” computations

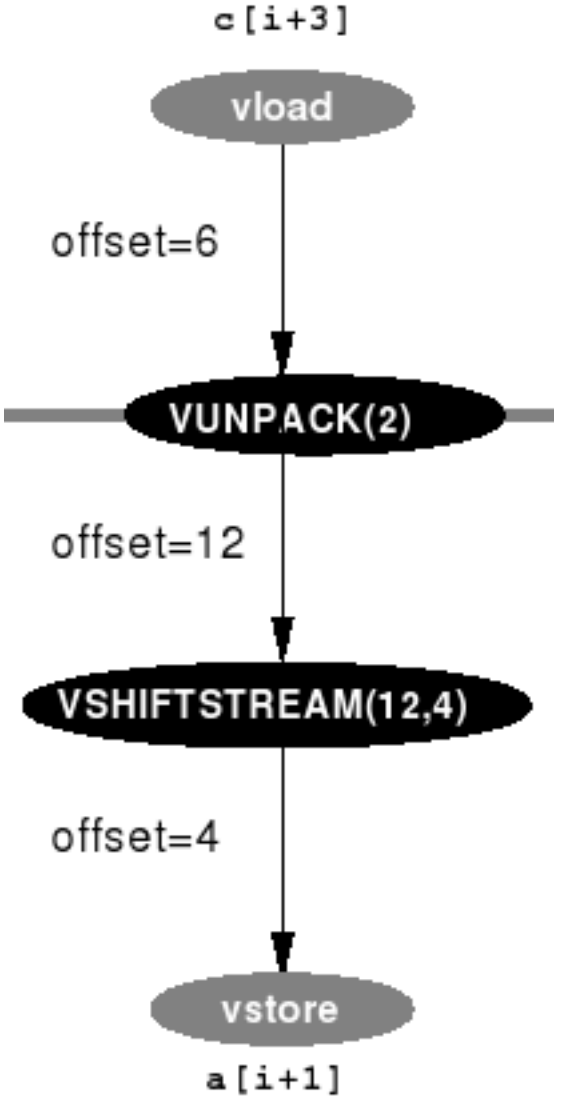
- Basic block-level
  - Seed virtual vectors
- “Short” loop-level
  - Unroll static loops
- “Loop”-level
  - Block (partially unroll) dynamic loops

Problems

- Strided access
- Alignment constraints
- Length/type conversion effects
- Compile-time knowledge
- Tension with ILP

Data Reorganization Graph

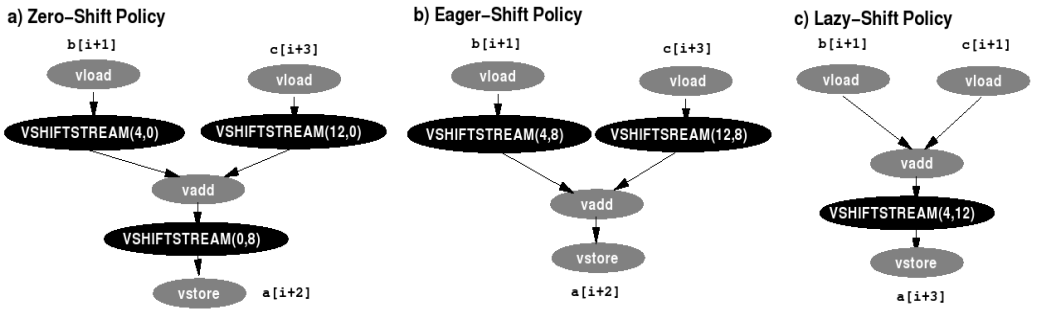
- Tree of vector expressions
  - Leaves: stream loads
  - Interior nodes: stream operations
    - vector ops
    - pack/unpack
    - stream shift
  - Root: stream store



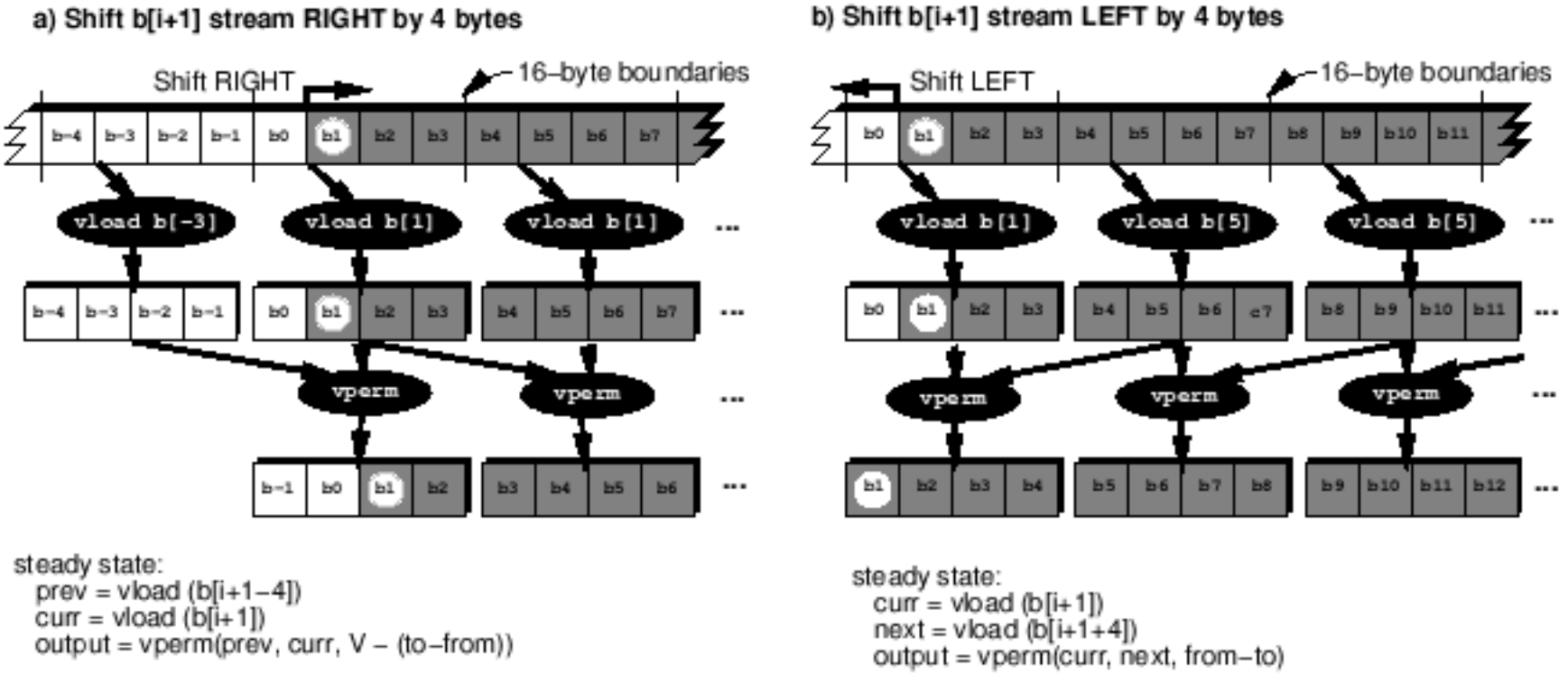
- Transformations
  - Goal: minimize instruction count
  - Alignment, type conversion, simplification, ...

Stream Shifting Policies

- (Zero):
  - Shift every load to offset zero
  - Shift every store to target offset
- Eager:
  - Shift every load to target offset
- Lazy:
  - Shift to target offset as late as possible
- (Dominant):
  - Shift intermediate expressions to dominant offset
  - Shift result to target offset



Basic Alignment



- Load from register-aligned memory
- Different left / right shifting code
- Forces only zero-shift for runtime alignment

Improved Alignment

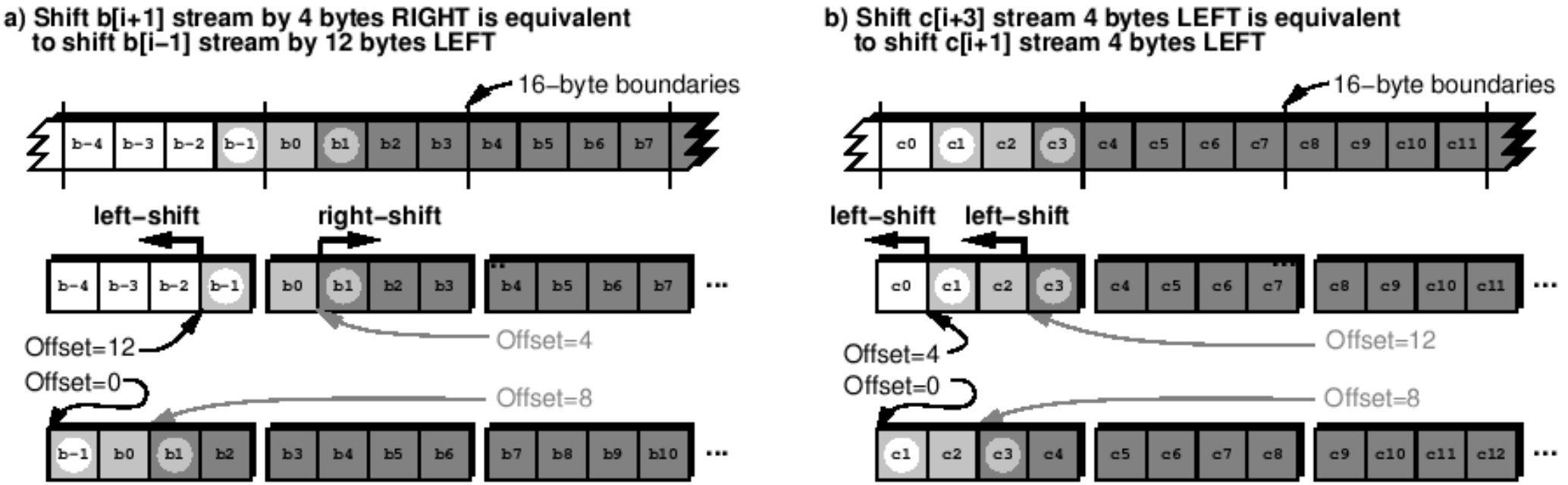


Figure 7. Converting arbitrary stream shift to an equivalent stream left-shift.

- Make everything into a left shift
- Prepend placeholder values and shift those to 0
- Allows any runtime policy

Length Conversion

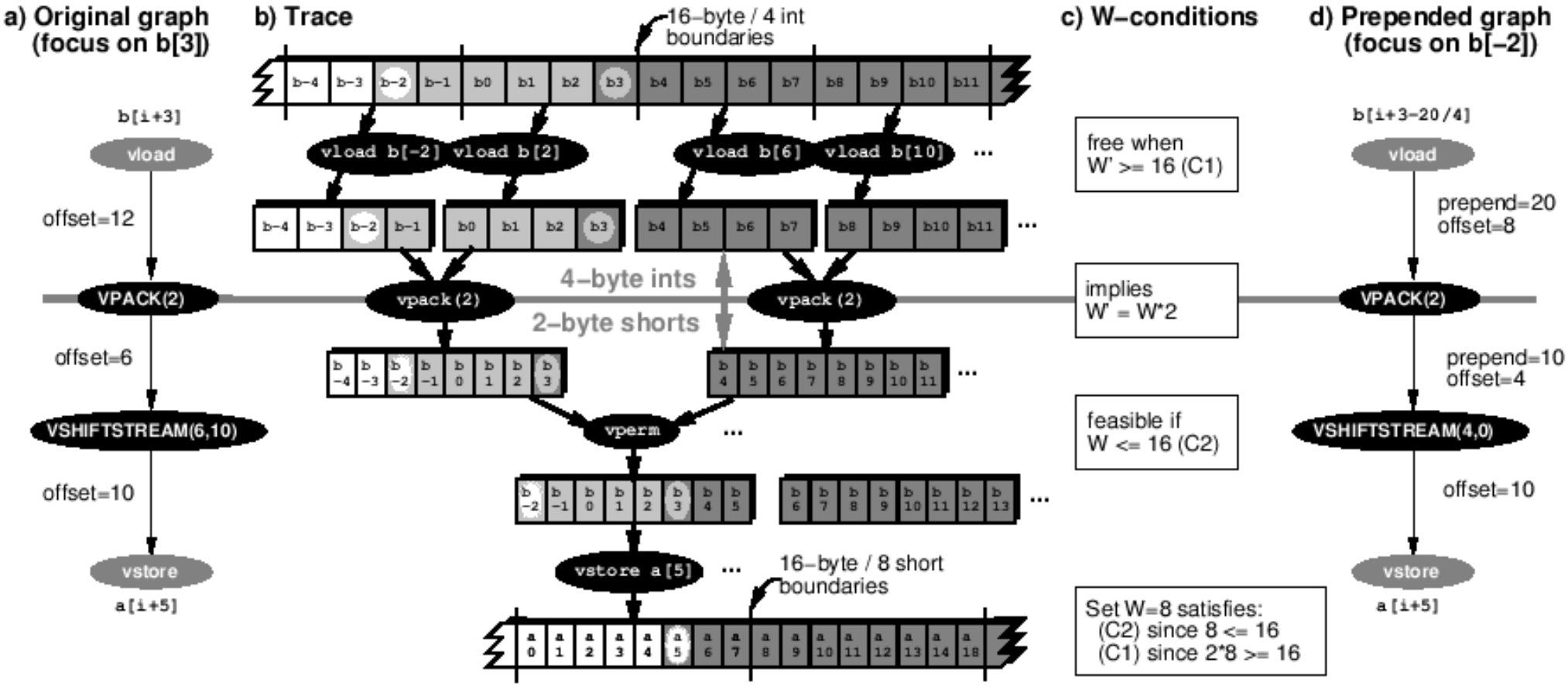


Figure 9. Conversion and shifting streams.

Length Conversion

- System has real hardware vector size V
- Create “virtual vector size” W and scale it across Un/Packs
- Problems:
  - ShiftStream only works if W <= V
  - Loading requires an extra shift if W < V

Devirtualization/Code Generation

- Select SIMD/scalar intrinsics
  - “Mixed-mode simdization”
  - Replace (un)pack, shift, and generic vector ops
  - Special case stores

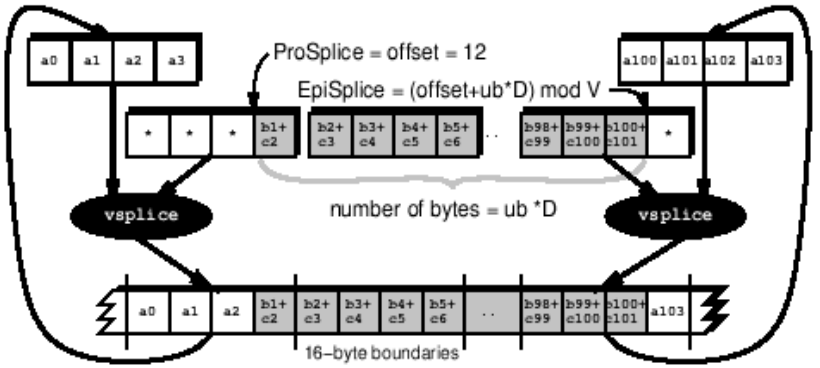


Figure 8: Special cases for prologue and epilogue.

- Balance DLP/ILP
  - Heuristically evaluate local decisions
  - Revert SIMD to scalar code where cheaper

Simdization Overview

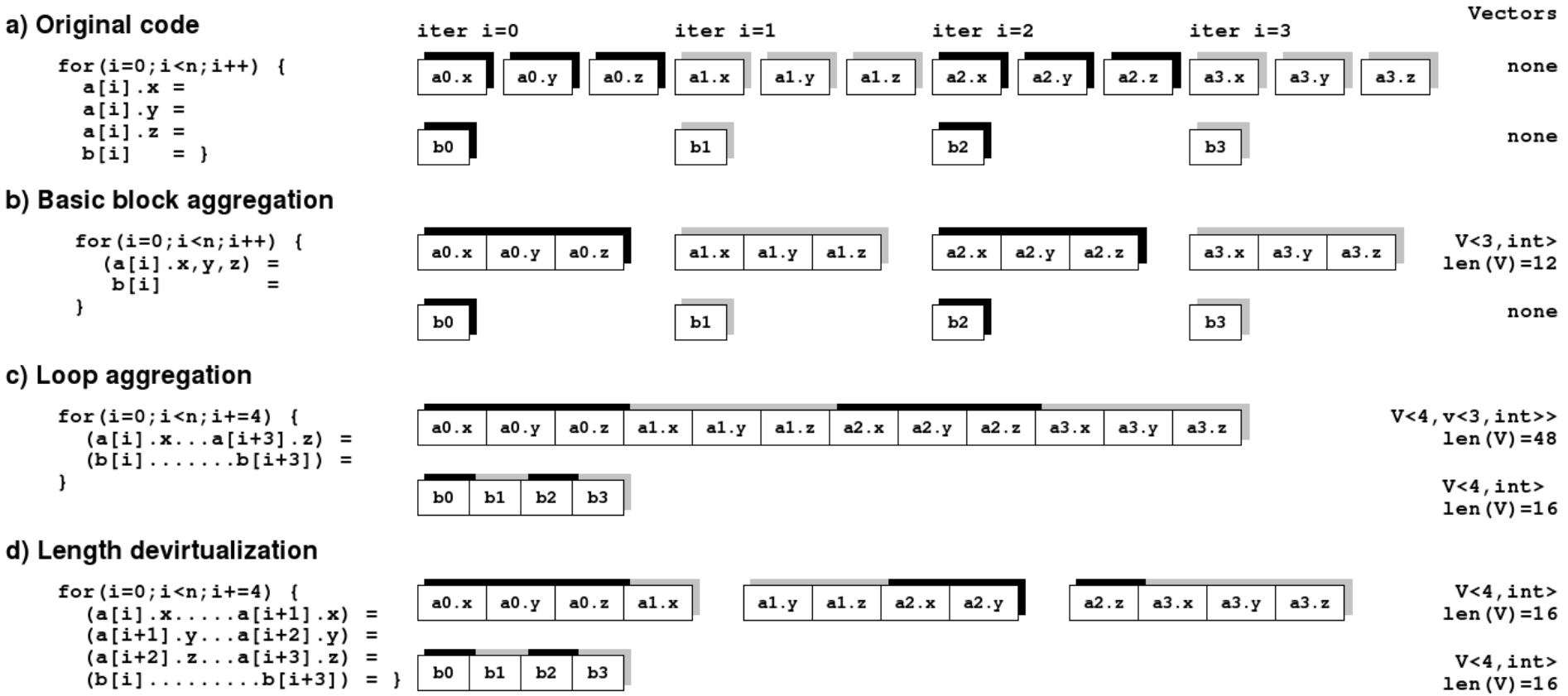


Figure 3: Mixed stride-one and adjacent accesses.

Questions?

Thank you!

Backup: Performance Impact

- Speedup: (oracle shift, actual) vs. scalar code
  - numerical.saxpy: (2.24, 1.08)
  - numerical.swim: (_, 1.38)
  - tcp/ip.checksum: (3.13, 2.92)
  - video.alphablending: (8.25, 6.14)
  - linpack: (_, 1.41)
  - Autocor: (_, 2.16)

Backup: Benchmark Results