# Vector-Based Acceleration in the IBM PowerEN™ Processor to Enable Software Defined Radio

Jeff H. Derby
IBM Research, RTP, NC
jhderby@us.ibm.com

Co-authors: Timothy Heil, Michele Franceschini, Anil Krishna, Bob Montoye, Dheeraj Sreedhar, Augusto Vega, Hangu Yeo, Charlie Johnson

# Overview

## We propose a platform as follows:

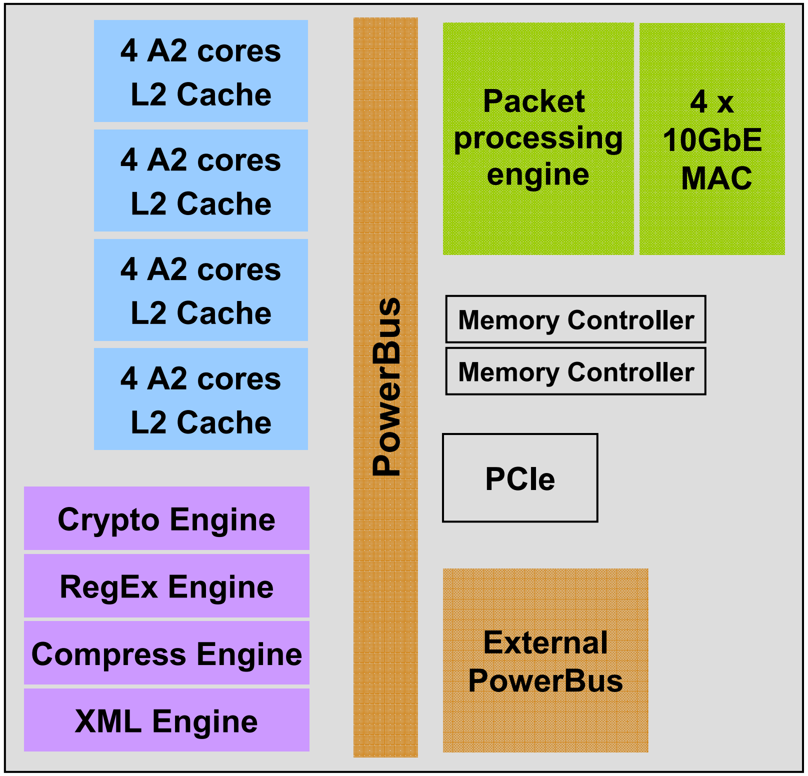
- derived from PowerEN, enhanced with vector-based acceleration (VBA)
- capable of supporting software-defined radio in maximally configured, macrocell wireless basestations
- with “in-line” acceleration
  - DMAs to / from hardware accelerators avoided
  - minimal data movement
- using an essentially traditional (general-purpose) programming model
- and an essentially general-purpose processor platform
- with a bus / memory subsystem employing hardware-managed coherency

# Outline

- PowerEN overview
- Vector-based acceleration
  - in the context of an enhanced PowerEN
  - architecture
  - programming model
- Algorithm examples
- A possible LTE-advanced application

# IBM PowerEN™ Processor System on a Chip

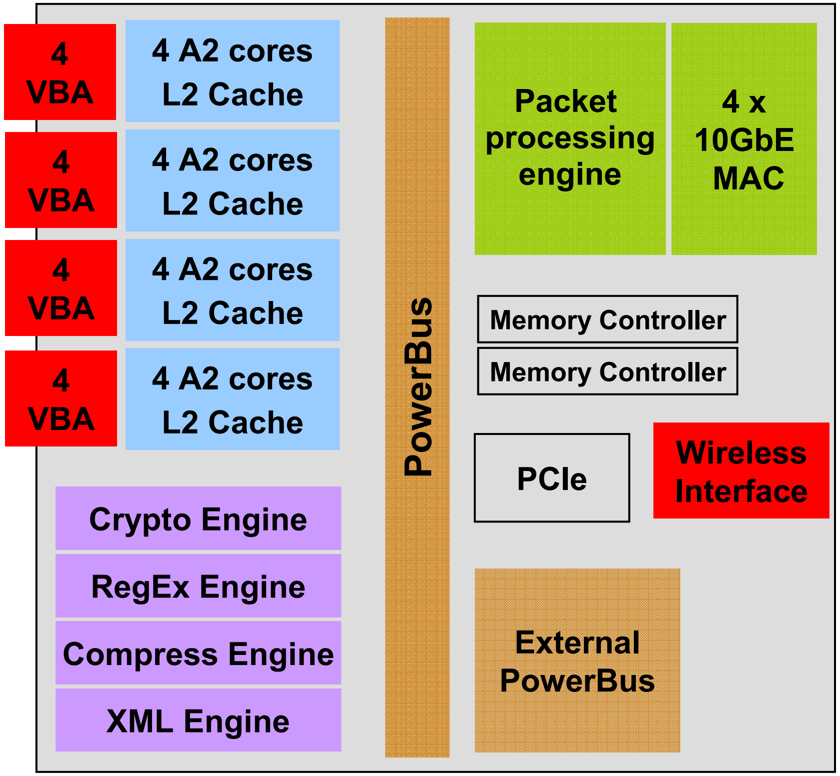
- Four At Chiplets
  - Four A2 cores per chiplet, 4 threads per core, 64 threads per chip
  - 2 MB shared eDRAM L2 per chiplet (8MB L2 / chip)
  - 64B cacheline
  - 2.3GHz operation
- Two Memory Controllers
  - Direct attach (UDIMM, RDIMM)
  - Each MC has two 72b DDR3 direct attach channels
- Acceleration Engines
  - PBIC attach with DMA engine
  - Compression / Decompression
  - Cryptographic co-processor
  - XML engine (XML transformation)
  - Regular Expression / Pattern-matching
- PowerBus
  - On chip coherent system bus
  - 1.75 GHz operation
  - One command bus
  - Four 16B data busses
  - “All peers” architecture
- 45nm, $410mm^2$




- Targeted at network-edge applications
  - intrusion detection / deep packet inspection
  - security / crypto acceleration
  - XML parsing / schema validation / ...
  - “smarter planet” solutions

# IBM PowerEN™ Processor System on a Chip

- Four At Chiplets
  - Four A2 cores per chiplet, 4 threads per core, 64 threads per chip
  - 2 MB shared eDRAM L2 per chiplet (8MB L2 / chip)
  - 64B cacheline
  - 2.3GHz operation
- Two Memory Controllers
  - Direct attach (UDIMM, RDIMM)
  - Each MC has two 72b DDR3 direct attach channels
- Acceleration Engines
  - PBIC attach with DMA engine
  - Compression / Decompression
  - Cryptographic co-processor
  - XML engine (XML transformation)
  - Regular Expression / Pattern-matching
- PowerBus
  - On chip coherent system bus
  - 1.75 GHz operation
  - One command bus
  - Four 16B data busses
  - “All peers” architecture
- 45nm, 410mm$^2$




- Targeted at network-edge applications
  - intrusion detection / deep packet inspection
  - security / crypto acceleration
  - XML parsing / schema validation / ...
  - “smarter planet” solutions

# Vector-Based Acceleration (VBA)

- A SIMD auxiliary execution unit (AXU)
  - can be attached to A2 cores (one per core) in an enhanced PowerEN
  - fed by the A2 core's instruction stream
- Based on VMX (aka AltiVec™) extended to 32-byte width
  - 8-wide for 32-bit fullwords, 16-wide for 16-bit halfwords, 32-wide for bytes
  - fixed-point and single-precision floating-point
- Includes:
  - native support for fixed-point complex arithmetic
  - special instructions for correlation with complex bit-vectors (e.g. for despreading)
- Key feature:  a very large, fully architected register file (VSRF)
  - 2048 256-bit registers (so 64KB total storage)
  - cache-line moves between the VSRF and the L2 cache
  - all accesses to the VSRF are via indirection using "map registers"
  - map registers contain pointers to data in the VSRF (registers / bytes / bits)
  - map register entries managed by software in SIMD fashion
- Capable of incorporating encapsulated special functions
  - example:  turbo decoder

# Programming Model Overview

## VBA uses an essentially “general-purpose” programming model

- load / store
  - no DMAs
  - but cache-line loads & stores can be used
- full system memory accessible via load / store
  - system-wide hardware-managed coherency

## but data can often be kept local to a VBA

- intermediate results kept in the VSRF
- function results may be kept in the VSRF
  - e.g. as inputs to the next function in a sequence
- function inputs may already be in the VSRF

## Net impact includes:

- (far) fewer memory accesses
- (significantly) reduced sensitivity to memory-subsystem inefficiencies

# VBA Indirection Architecture

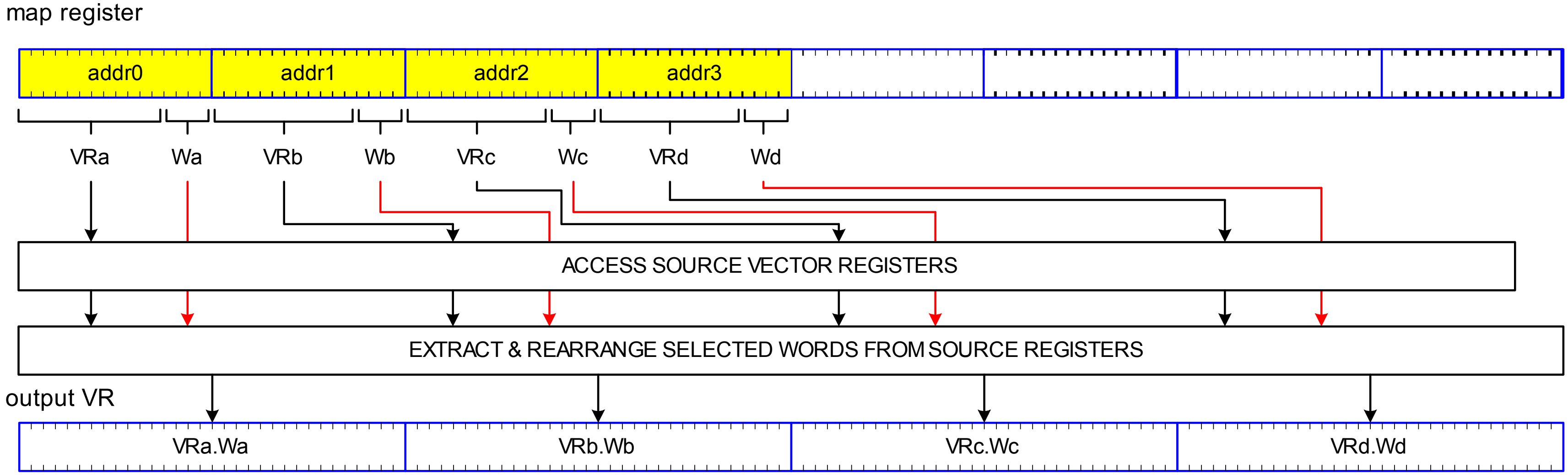
## Capabilities provided by indirect access to the VSRF:

- Specify one of 2048 archtected registers in a 5-bit register operand field
  - compatibility with existing PowerPC instruction formats
- Dynamic addressability of data elements in the VSRF
  - vectors, words, bytes, bits
  - data elements in the VSRF can be accessed (and indexed) as if in memory
  - addressed data elements can be variable-length

## The indirection mechanism supports:

- “Operand-associated” indirection
  - first 16 map registers used as four 32-entry maps, one per register-operand
  - enables naming one of 2048 (or more) registers in a 5-bit field
- “Generalized” indirection
  - gather up to eight data elements from arbitrary locations in the VSRF to a single register
  - move / copy a data element between arbitrary locations in the VSRF

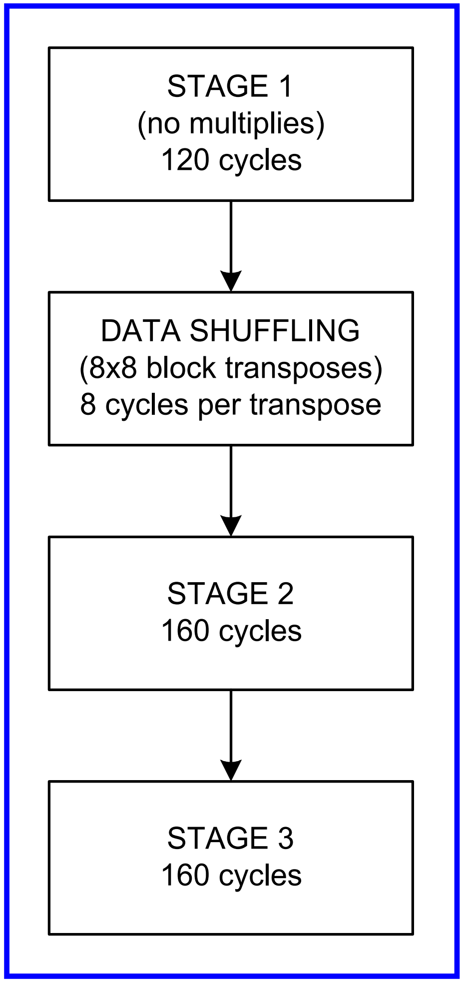
# Generalized Indirection Example: Gather Words



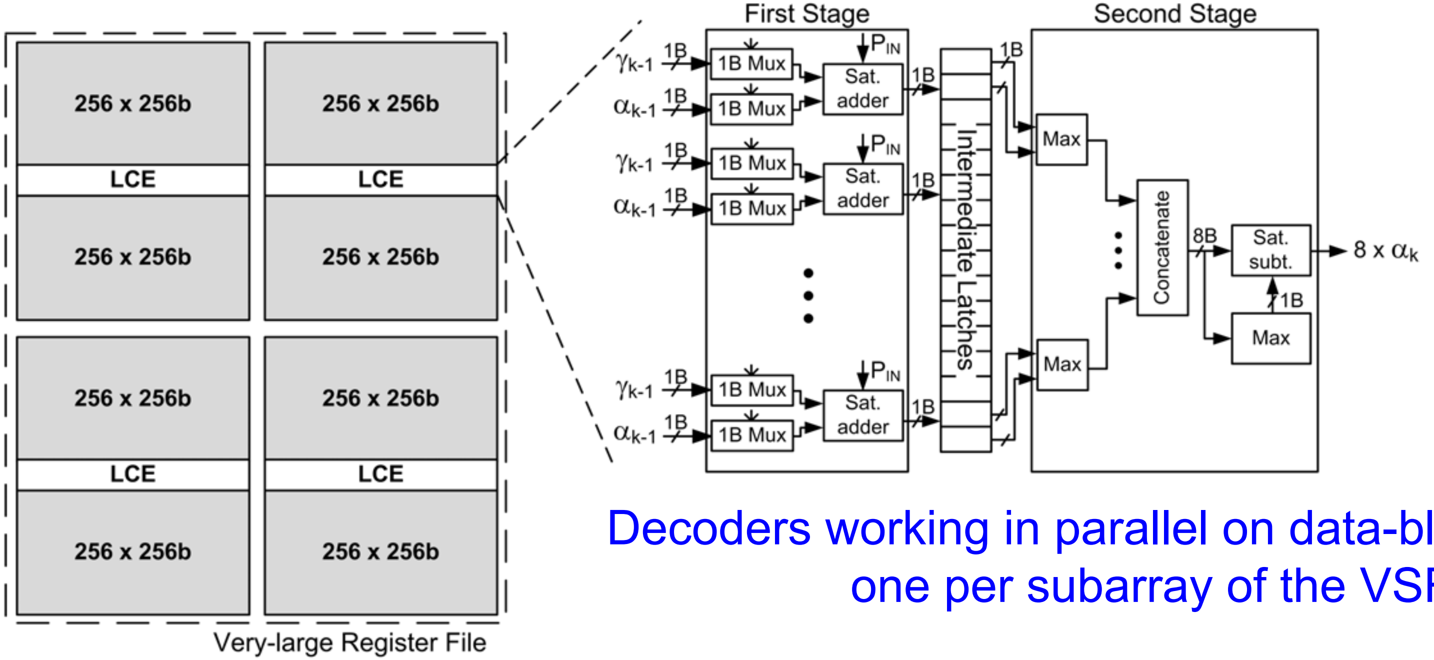

- uses up to 8 map-register entries per operation (four leftmost entries in this example)
- addr0 through addr3 are byte offsets in the reg file of desired word elements
- each address decodes to:
  - a register index in the reg file (e.g. VRa from addr0)
  - a byte offset of the desired word in the register (e.g. Wa from addr0)
- selected words are ordered in the target register in the reg file per ordering of pointers in the map register
  - rightmost 4 entries in the target register filled based on rightmost 4 entries in the map register
- data-element lengths (words, in this example) implicit in instruction opcode

# Algorithm Example – 512-point FFT

- Fixed-point
  - data and twiddles are 16-bits real, 16-bits imag
- Algorithm structure is radix-8 DIT
  - radix-8 is a "perfect match" for 8-wide SIMD
  - VBA is 8-wide SIMD for (16,16)
  - 8 "radix-8 butterflies" in parallel in the 8-wide SIMD
  - net 3 stages plus one data-shuffling step
- Memory accesses:
  - data and twiddles loaded at the outset
  - result stored at the end
  - all intermediate results kept in the VSRF
- Managing accesses to intermediate results
  - done entirely through management of map-register entries
  - ➢ as if the intermediate results were in memory
- And:
  - what if the input data were already in the VSRF?
  - what if the output data can be used directly from the VSRF?
- Larger FFTs:
  - if the size is divisible by 8, start with radix-8 and only one data-shuffling stage is needed

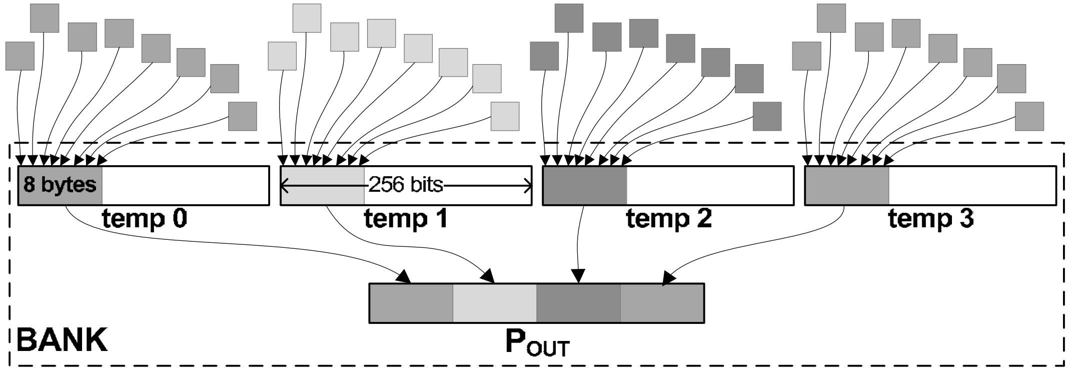


# Algorithm Example – Turbo Decoder




Decoders working in parallel on data-block partitions, one per subarray of the VSRF

De/interleave using gather capability within the VSRF

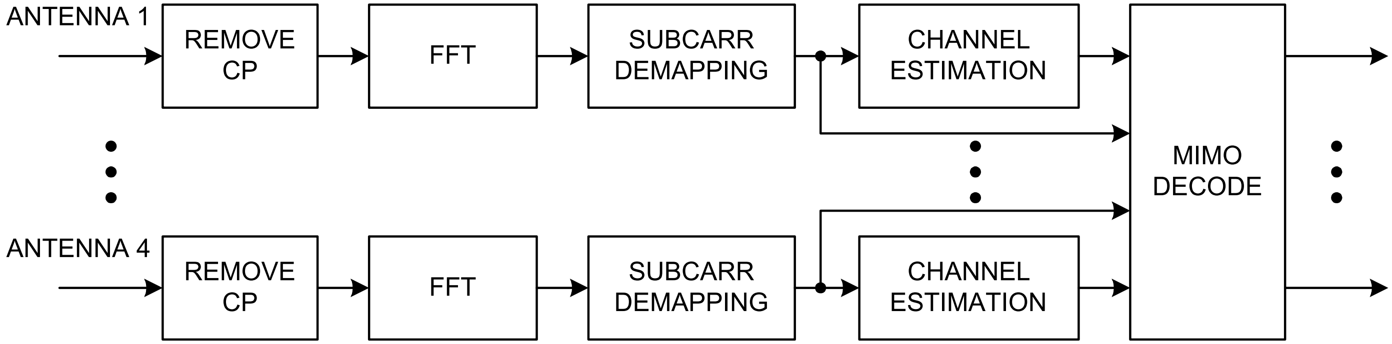


# Algorithm Performance Projections

| ALGORITHM | EXECUTIONS / SEC or THROUGHPUT | COMMENTS |
|---|---|---|
| FFT, 512-point | 4.4E6 / sec | radix-8<br>data movement not included |
| FFT, 1K-point | 2.0E6 / sec | radix-8 followed by radix-2<br>data movement not included |
| FFT, 2K-point | 920k / sec | radix-8 followed by radix-4<br>data movement not included |
| MATRIX INVERSION, 4x4 | 115E6 / sec | implemented in floating-point<br>conversion to/from float included in perf. projection |
| TURBO DECODER | 230 Mbits / sec | 6 iterations<br>using local computation elements embedded in the VSRF |

- All projections are for a single A2-like core with one VBA unit operating at 2.3GHz

# An LTE-Advanced Configuration: start with uplink the "front-end"




per 20MHz channel, per sector:
per 500µsec slot for 4x4 MIMO:
28 2K-point FFTs → < 40µsec
8400 4x4 inverts → about 140µsec

Note also: For each symbol, the received antenna data can remain in the VSRF as it is processed through demapping; and all data for the 4x4 MIMO can remain local in a node's L2

Overall projections for complete LTE digital baseband including turbo decoder:
4x4 MIMO, one 20MHz channel, one sector: less than 3 A2+VBA cores (about 2.5)
4x4 MIMO, three sectors, two 20MHz channels per sector: about 16 A2+VBA cores

# Summary

## We have propose a platform as follows:

- derived from PowerEN, enhanced with vector-based acceleration
- capable of supporting software-defined radio in maximally configured, macrocell wireless basestations
- with “in-line” acceleration
  - DMAs to / from hardware accelerators avoided
  - minimal data movement
- using an essentially traditional (general-purpose) programming model
- and an essentially general-purpose processor platform
- with a bus / memory subsystem employing hardware-managed coherency

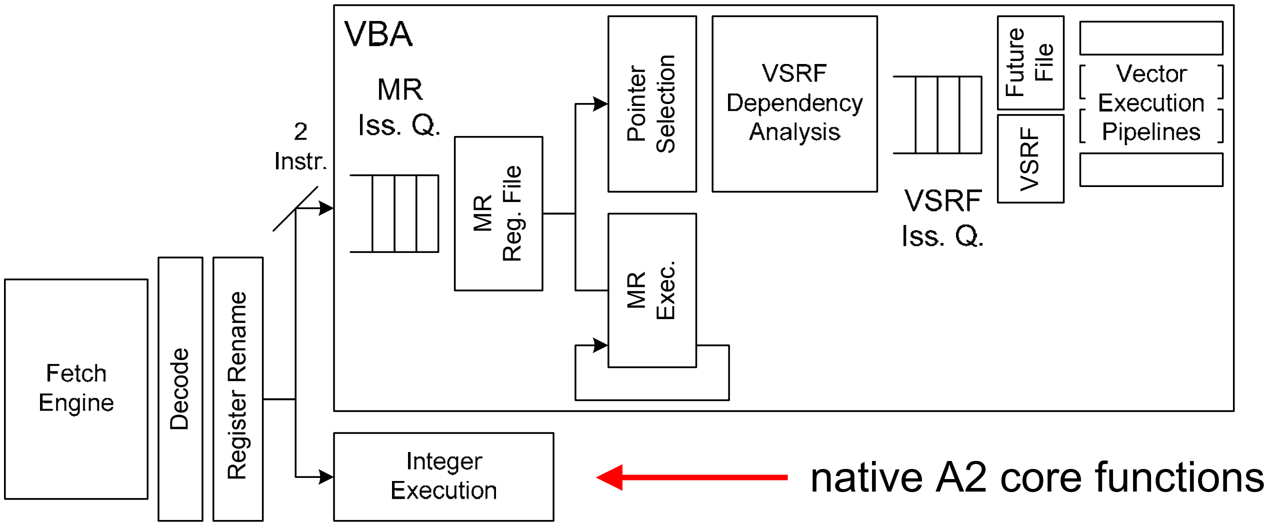
# Acknowledgments


- In addition to the listed co-authors, there were many who contributed to the work reported, including:
  - Brian Rogers
  - Steve VanderWiel
  - Jason Cantin
  - Russ Hoover
  - Chuck Cox
  - Matt Tubbs
  - Scott Higdon
  - Nadav Levison
  - Erez Barak
  - Ayal Zaks
  - Mircea Namolaru
  - Revital Eres
  - Sagi Manole
  - Alejandro Rico

# Thank You

# Backup

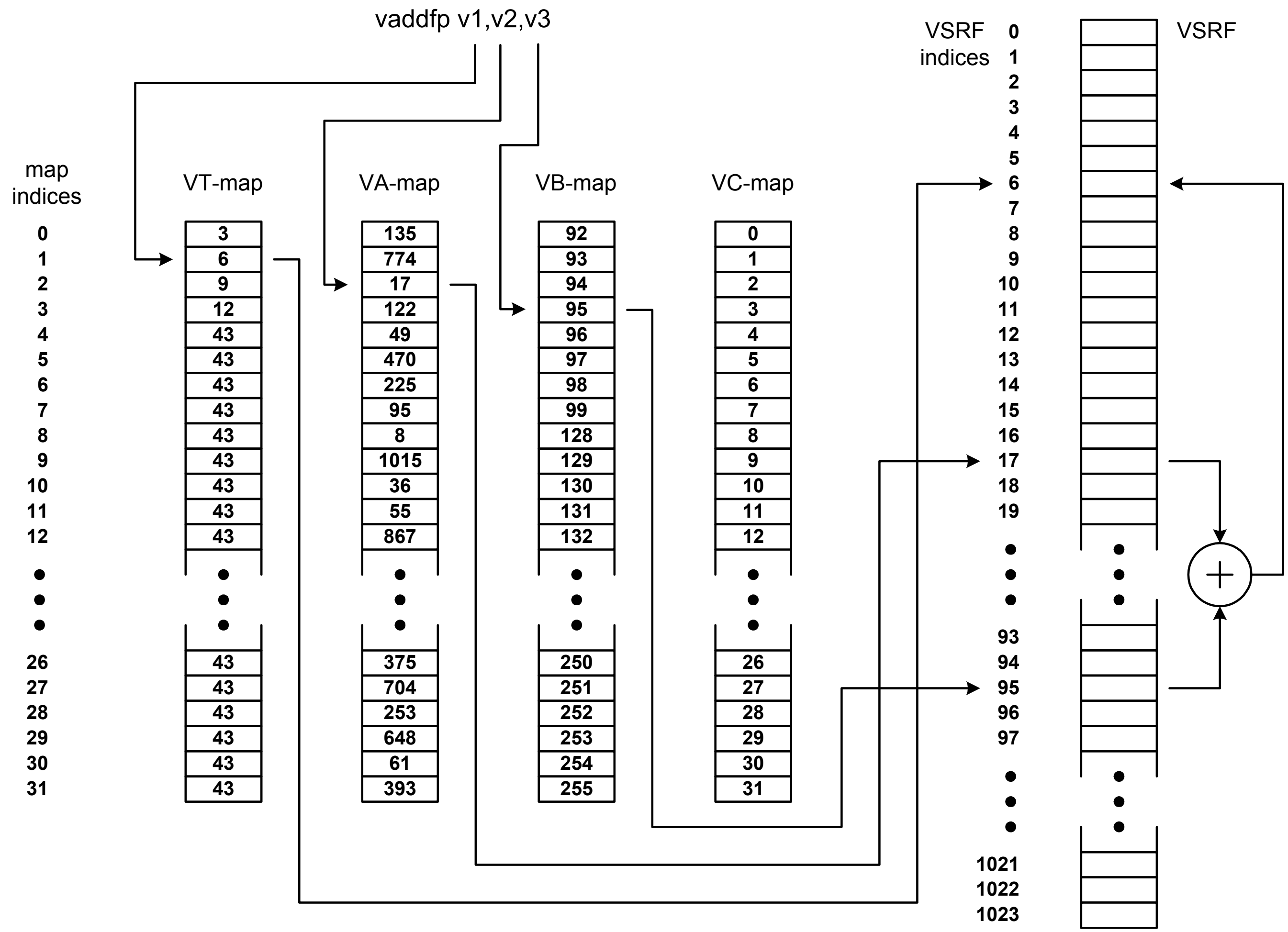
# VBA Microarchitecture




- VBA as an AXU attached to an “A2-like” core
- can sustain two instructions per cycle through the VBA unit
- native core provides:
  - scalar integer functions
  - load / store
  - branch
- overall microarchitecture is “traditional superscalar”

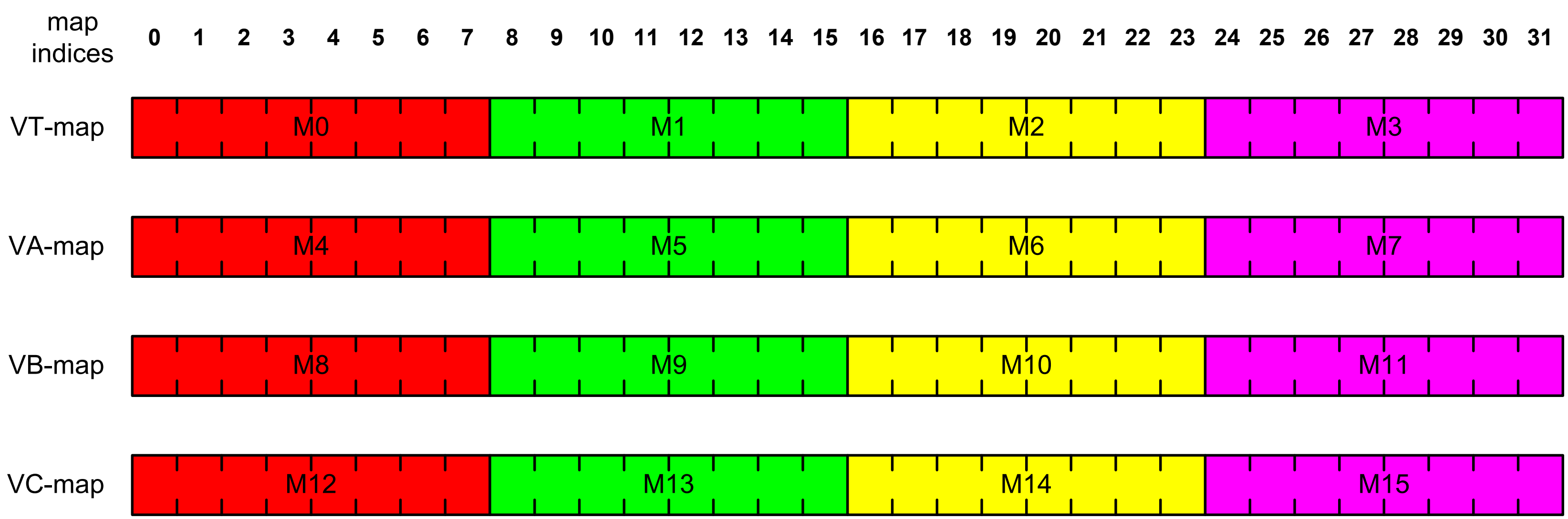
# Operand-Associated Indirection – An Example

VT map:
map regs 0 - 3
VA map:
map regs 4 – 7
VB map:
map regs 8 – 11
VC map:
map regs 12 - 15




➢ Note: In this example, map entries are shown as register indices
(in fact, map entries are byte offsets from the origin of the VSRF)

# Map Registers for Operand-Associated Indirection




- 32 map registers altogether
  - M0-M15 used for map-based indirection
  - formatted as shown (8 map entries per map register, one entry per halfword)
  - map entries use 16 bits each (VSRF architected limit → 64KB)
- Map registers “look like” VMX registers:  8 halfwords per 128-bit register
- SIMD orientation of map management
  - operations on map registers, not on individual map entries

# "Vector String Register File" (VSRF)

## Organization:

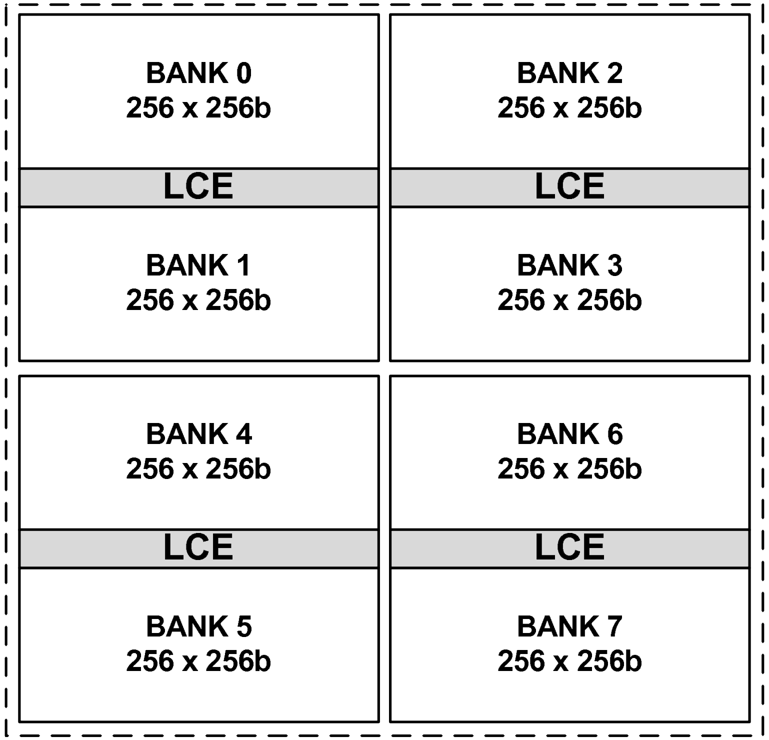
- 8 banks of 256 256-bit regsters
- each bank has four read ports, one write port

## Physical structure:

- 10T cells
- 2R,1W per cell, double-pumped read
  - so 4R,1W per cell
- area about 50% of the net VBA area

## Use of "local computation elements":

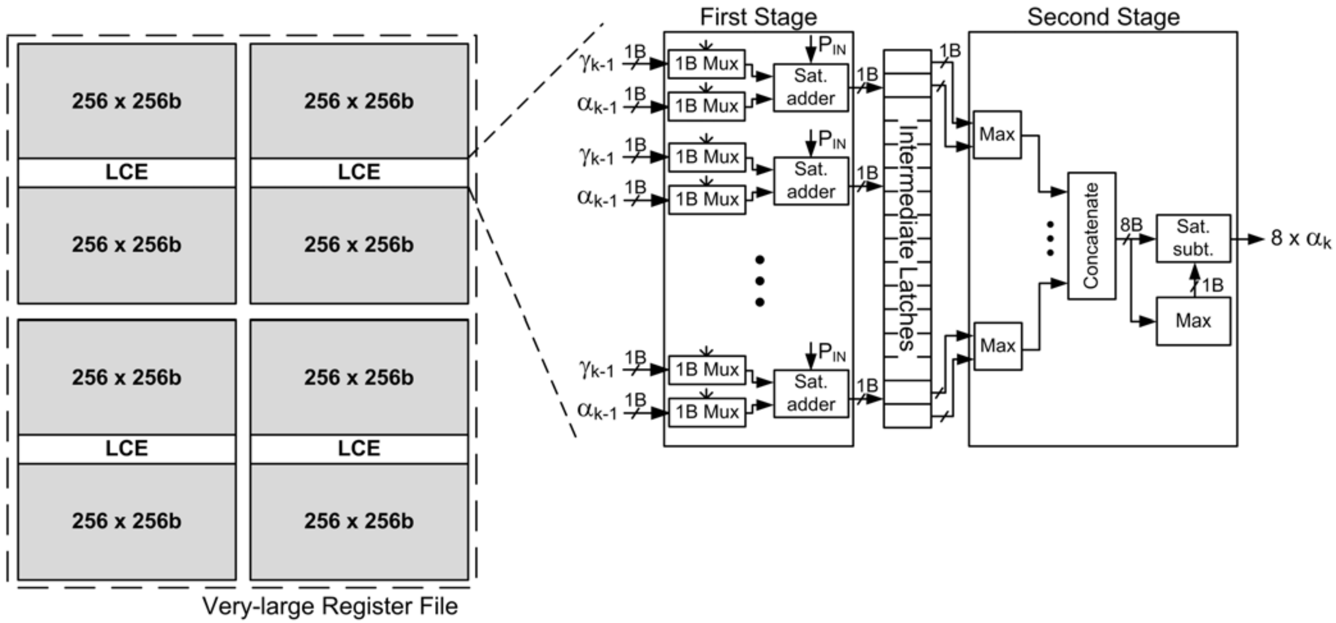
- logic embedded in the register file
  - viable if the logic is relatively simple
- leverages full available parallelism
  - 4 read ports & 1 write port per bank
- used for "shrink-wrapped" turbo decoder capability



# Modeling and Assessment

- Algorithms are compiled from C or hand-coded in assembler
  - vectorization is by hand
  - extensions to gcc being developed to target VBA
- Code runs on functional simulator
  - runs on a simulated "real machine running an OS"
  - functional correctness of the code can be verified
- Instruction traces generated by the functional simulator
  - traces show all memory addresses accessed by instructions
  - traces show all relevant details of the VBA indirection mechanism
- Traces are run on a performance model
  - model includes all microarchitecture details of "A2 + VBA" given current design status

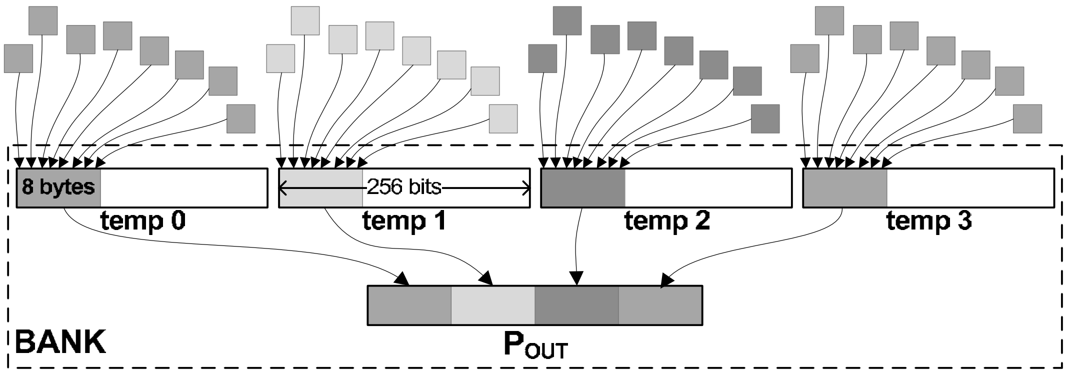
# Turbo Decoder – Decoding Stage




**Turbo Decoding parallelization:**

- The *codeword* is split into 8 chunks (the number of VSRF banks).
- Each bank locally decodes its assigned chunk, by making use of its attached LCE.
- Each LCE is shared by two banks: it is fed by its even bank on even cycles and by its odd bank on odd cycles.
- Each LCE incorporates the logic required for forward/backward recursion computation.
- All LCEs can be concurrently driven by special Turbo Decoding instructions added to the VBA ISA.

# Turbo Decoder – De/Interleaving Stage




**Data shuffling based on the VBA gather capabilities:**

- Extrinsic probabilities $P_{EXT}$ generated during decoding are shuffled based on mapping information stored in the VSRF.
  - The mapping information is loaded into the map registers before executing the gather instructions.
- Four groups of eight $P_{EXT}$ values each are first gathered into four temporal registers in the current bank.
- These four groups are then gathered into a single register, forming a set of 32 $P_{EXT}$ values.
- This process is repeated until all the $P_{EXT}$ values are moved to their final positions in the VSRF.