Supporting Vector Programming on a Bi-Endian Architecture

Bill Schmidt, Ph.D.

Linux on Power Toolchain Development

IBM Linux Technology Center

Michael Gschwind, Ph.D.

System Architecture

IBM Systems and Technology Group

Outline

- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

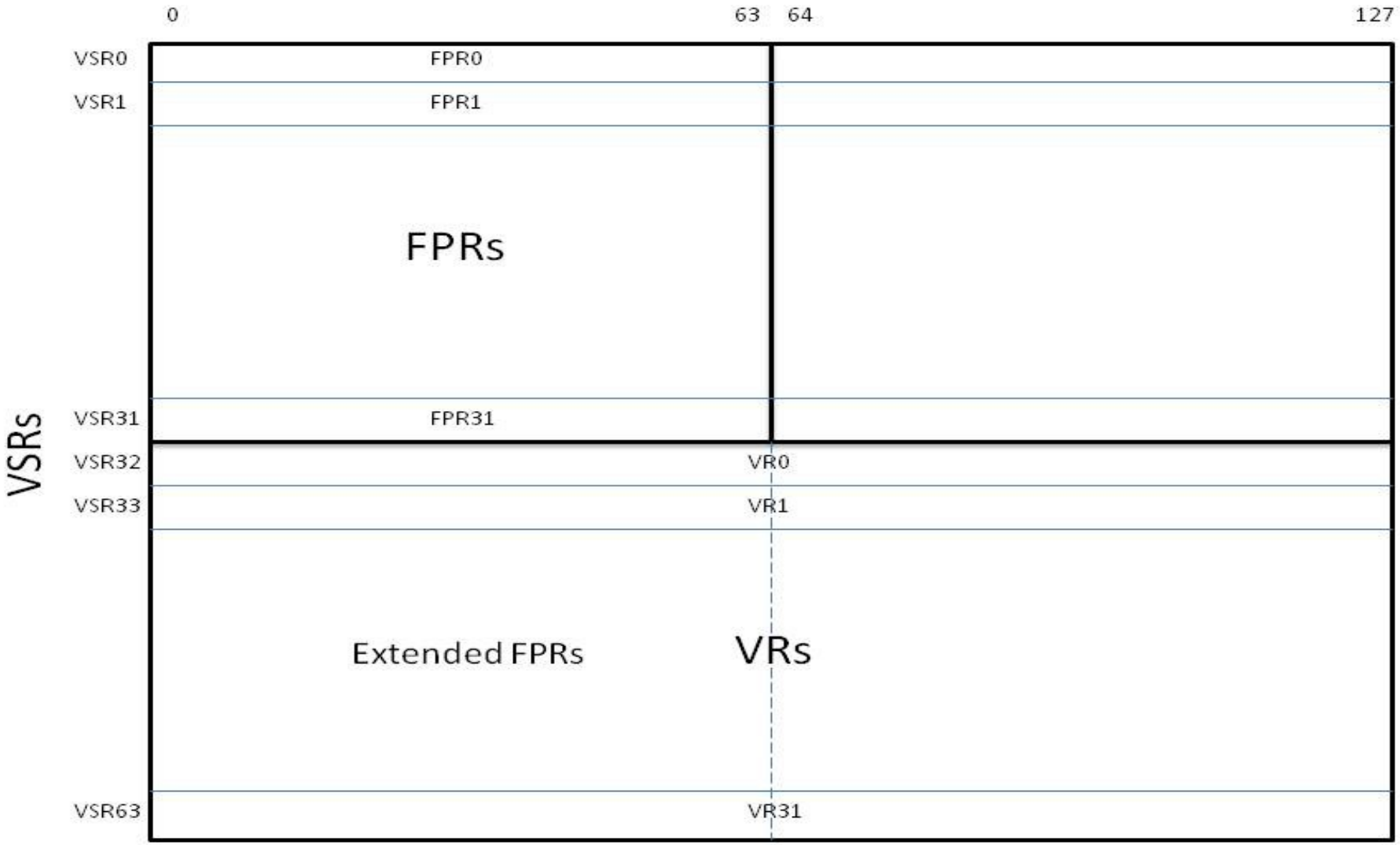
Outline

- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

History: Power ISA Vector Support

- Altivec (VMX) – Apple/IBM/Motorola alliance
  - 32 x 128-bit vector registers, aligned memory access only
  - Support for 16xi8, 8xi16, 4xi32, 4xf32, “bool” and “pixel” types
  - Power Mac G5, JS20 blade server (ISA 2.03, PPC970) – **2003**
  - POWER6 (ISA 2.05) - 2007
- VSX – Vector-scalar extensions
  - Extended vector register file to 64 x 128-bit, scalar float to 64 x 64-bit
  - Added support for 2xf64 and (limited) 2xi64
  - Unified floating-point and vector register files
  - Unaligned memory access supported but somewhat slow
  - POWER7 (ISA 2.06) - **2010**
- Expanded VSX and VMX instruction sets
  - Expanded 2xi64 ops, i128 ops, crypto, logical, decimal, i64/f64 VSX load/store, merge even/odd, ...)
  - Improved unaligned vector access performance
  - POWER8 (ISA 2.07) - **2014**

Unified register file



History: Power ISA Endianness

- The Power ISA has been bi-endian since its inception.
- All compliant processor implementations support an endian mode selectable by the supervisor/hypervisor.
- Server operating systems have generally been big endian.
  - Except: Short-lived implementations of Solaris and Windows NT
- Server program models for vector programming have been exclusively big-endian.
  - VMX contains (intentional or unconscious) big-endian “biases” (inherent byte and element numbering).
  - VSX adds some vector memory access instructions that contain intentional big-endian biases.

Outline

- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

Little Endian for Linux on Power

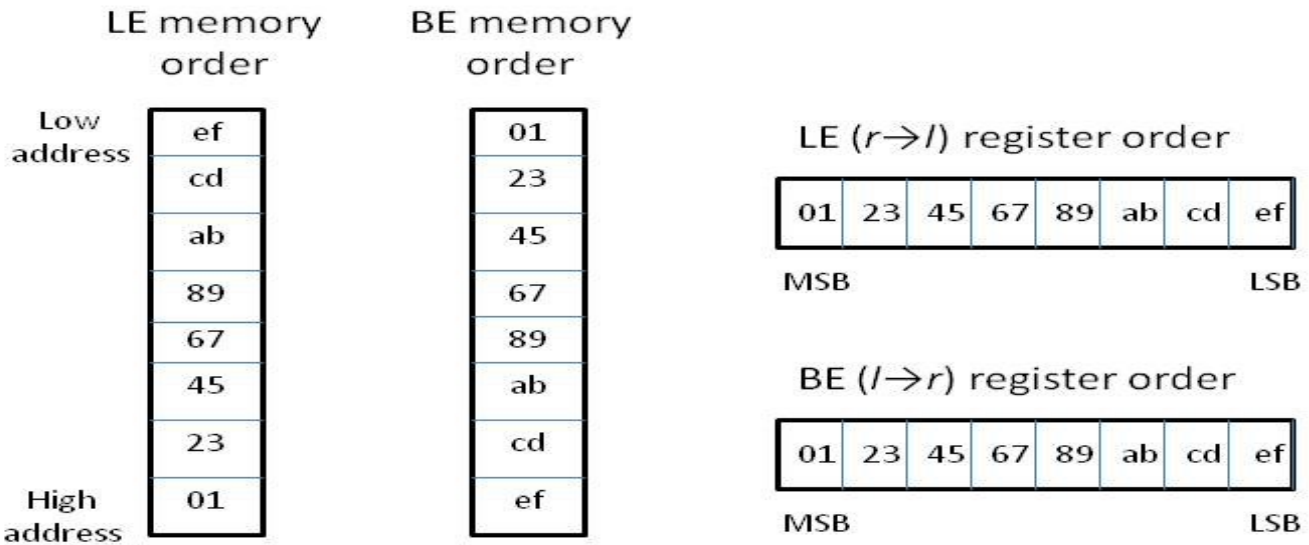
- Changes in technology and market forces make a Little Endian offering attractive.
  - Slower growth in single-thread CPU performance
    - Accelerator hardware becomes more attractive.
    - Existing GPGPUs use LE data layout.
  - Growth in Linux share of server market
    - Most Linux apps developed for LE systems.
    - Though porting from LE to BE is usually not difficult, the perception is otherwise – obstacle to “mindshare” – and there are exceptions.
- LE Linux on Power distribution plans
  - Ubuntu Server (14.04 *ff*.) – available now
  - SLES 12 – available now (10/27)
  - RHEL is planned (standard disclaimers...)
- All offerings are 64-bit only.

Outline

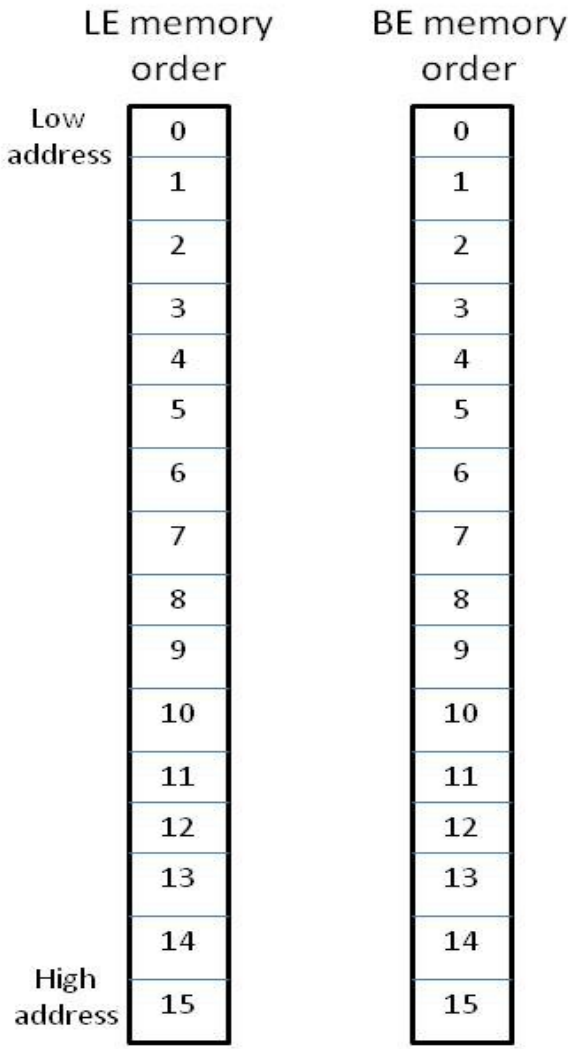
- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

Byte order and element order: Scalars

- Scalar values
  - BE: Most significant byte (MSB) stored at lowest memory address
  - LE: Least significant byte (LSB) stored at lowest memory address
  - In registers, both BE and LE look the same.



Byte order and element order: Byte arrays



LE ($r \rightarrow l$) register order

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

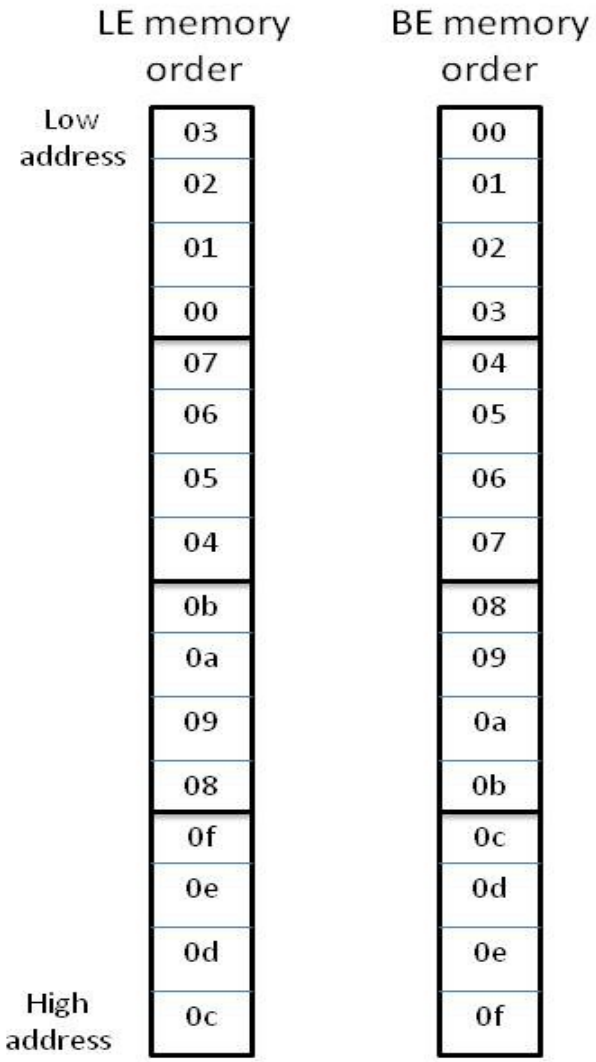
MSB LSB

BE ($l \rightarrow r$) register order

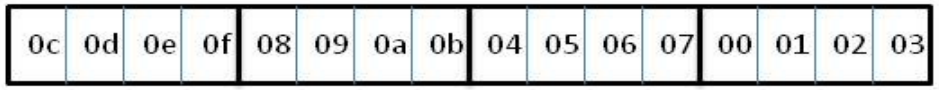
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

MSB LSB

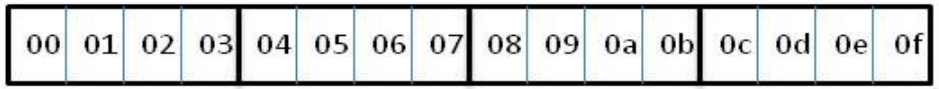
Byte order and element order: Word arrays



LE ($r \rightarrow l$) register order



BE ($l \rightarrow r$) register order



Outline

- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

Vector programming models: Existing

- Power already supports an extensive vector interface, accessed via <altivec.h>.
  - Many hundreds of vector intrinsics, overloaded by type

```
vector int va, vb, vc;
va = vec_add (vb, vc);
vector float vfa, vfb, vfc;
vfa = vec_add (vfb, vfc);
```

  - Each intrinsic maps to a single VMX or VSX instruction.
  - Supported by GCC, LLVM, and XL compilers on Linux, AIX, and other platforms
  - Uses l→r element order where ordering matters
  - Vectors map naturally on top of arrays.
- How should this model be modified for different consumers on little endian?

Vector programming models

- Conflicting goals
  - LE programmers expect r→l element ordering.
  - Existing open source compilers do also.
  - Experienced POWER programmers are used to l→r ordering.
    - BE libraries created with this assumption
  - Many VMX/VSX instructions encode l→r element numbers.
    - Explicitly: vspltw v2,v1,0 (duplicate zeroth word from the left)
    - Implicitly: merge-high/low, multiply even/odd
    - Baked into original Altivec design – but Altivec loads/stores respect endianness
- Two LE programming models provided
  - True-LE: LE memory order, r→l register element order
    - Chosen as default to ease porting of Linux applications
  - Big-on-Little (BoL): LE memory order, l→r register element order
    - Not discussed further today due to time constraints

Vector programming models: True-LE

- The True-LE programming model asserts that vectors in registers will use r->l element order.
  - Matches assumptions of GCC and LLVM
  - Natural fit for VMX loads and stores
- Many vector intrinsics are “pure SIMD.”
  - Computations operate within “lanes”
  - No changes necessary
- But
  - Implicit and explicit encoding of element numbers are wrong (l→r)
    - Intrinsics must fix these up.
    - No longer a 1-1 correspondence between intrinsics and instructions
  - VSX loads and stores produce “swapped” element order.
    - May be generated by compiler for assignments, not just intrinsics
    - Permute operations (“swaps”) are needed to adjust the ordering.

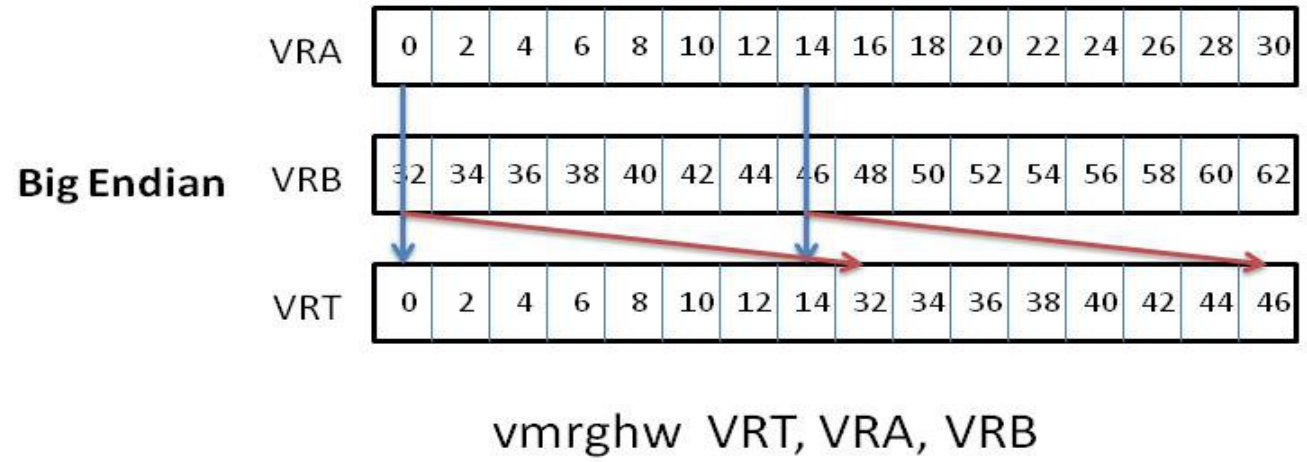
Vector programming models: True-LE

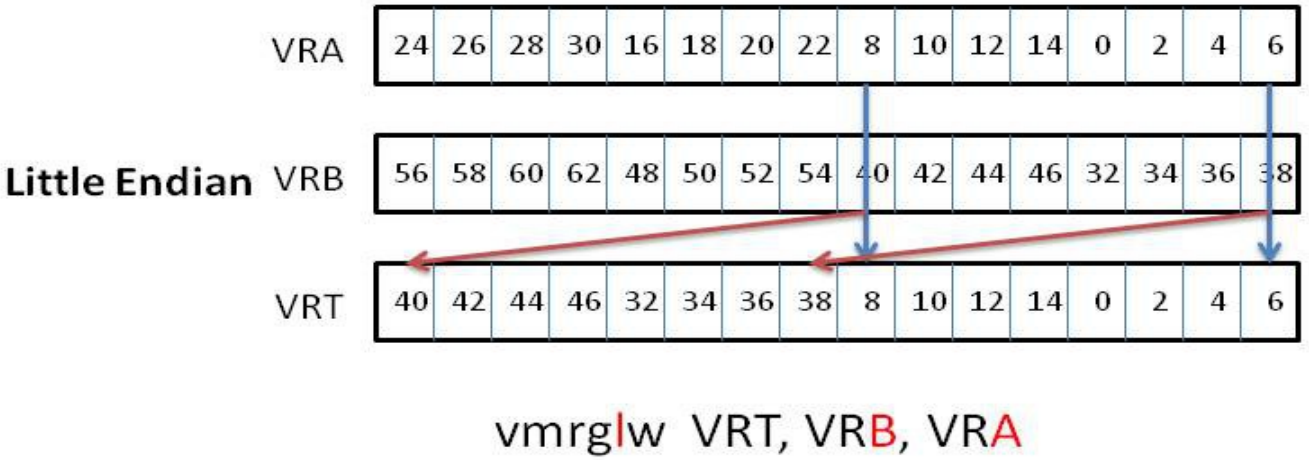
- The True-LE model is appropriate and useful to most consumers.
  - Apps ported from other LE platforms
    - Straightforward mapping from “old” interface to “new” one using the same element ordering assumptions
  - New apps
    - Programmers coming from an LE environment will find this natural; programmers familiar with the <altivec.h> interface can use it without (much) change.
  - Apps ported from BE Power
    - Most apps can be ported without (much) change.
    - However, apps using the VSX load/store intrinsics may require or prefer another approach.
- Vectors still map naturally onto arrays.

Outline

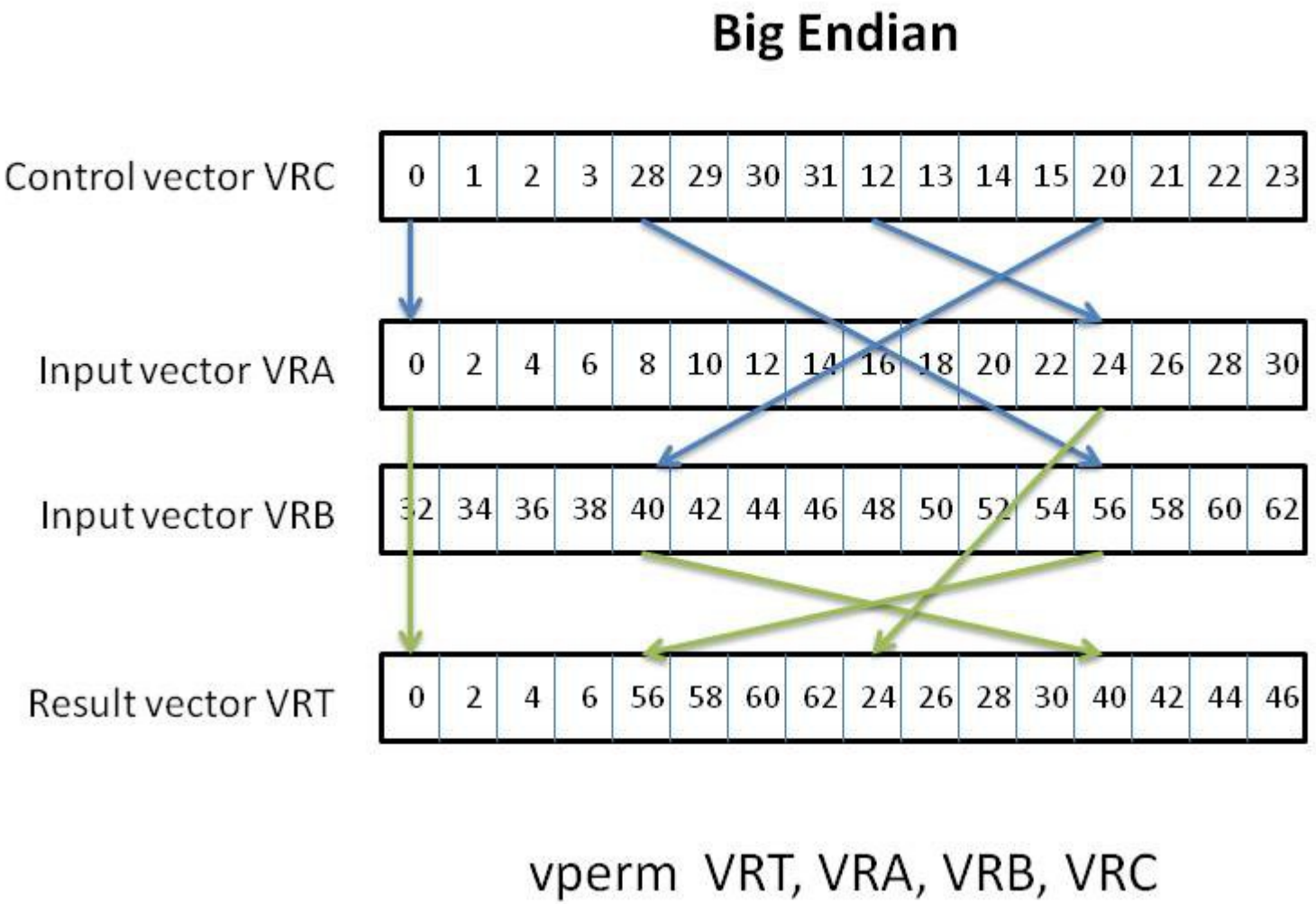
- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

Example: vec_mergeh

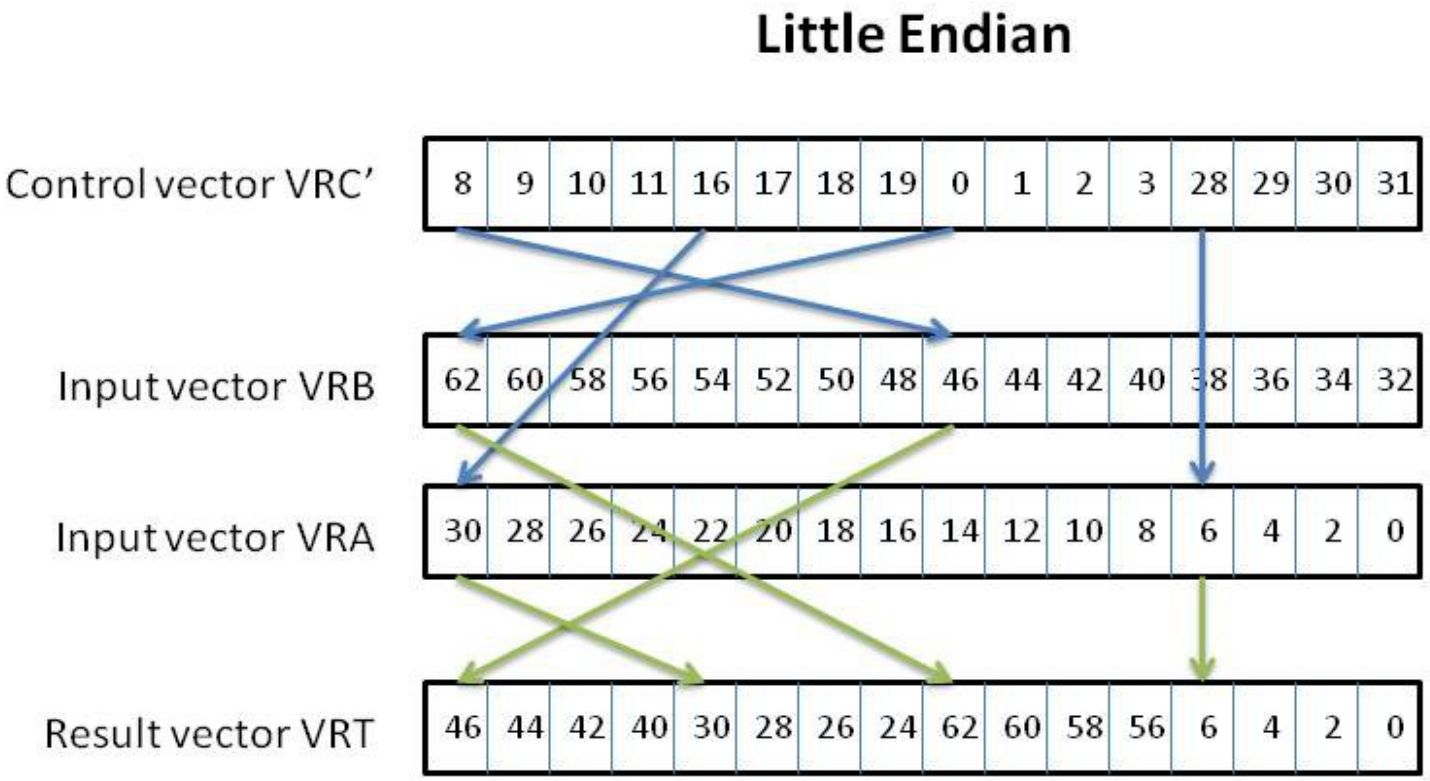




Example: vec_perm



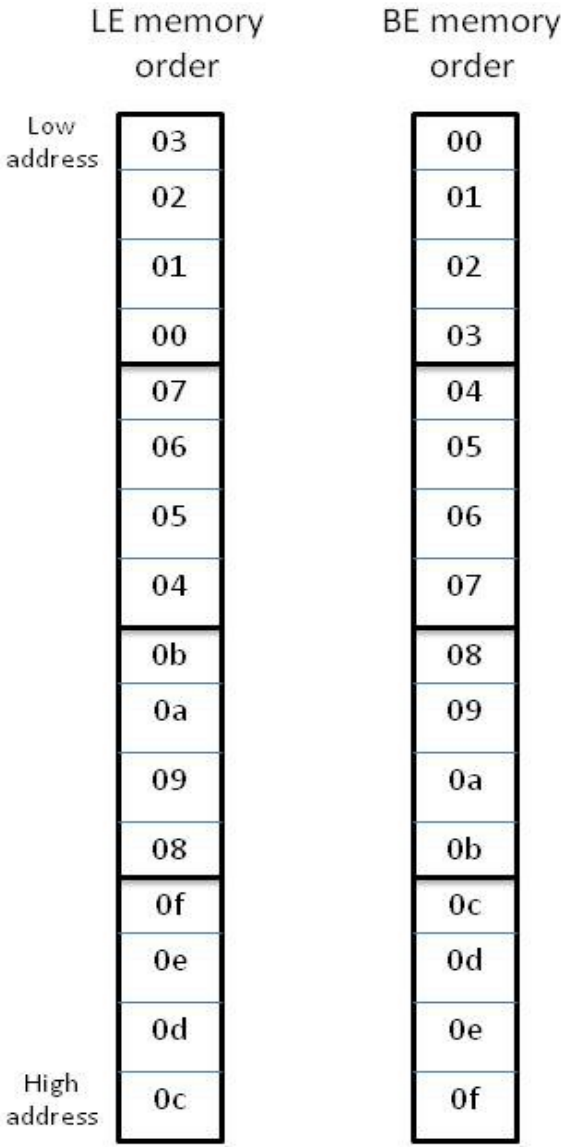
Example: vec_perm, cont.

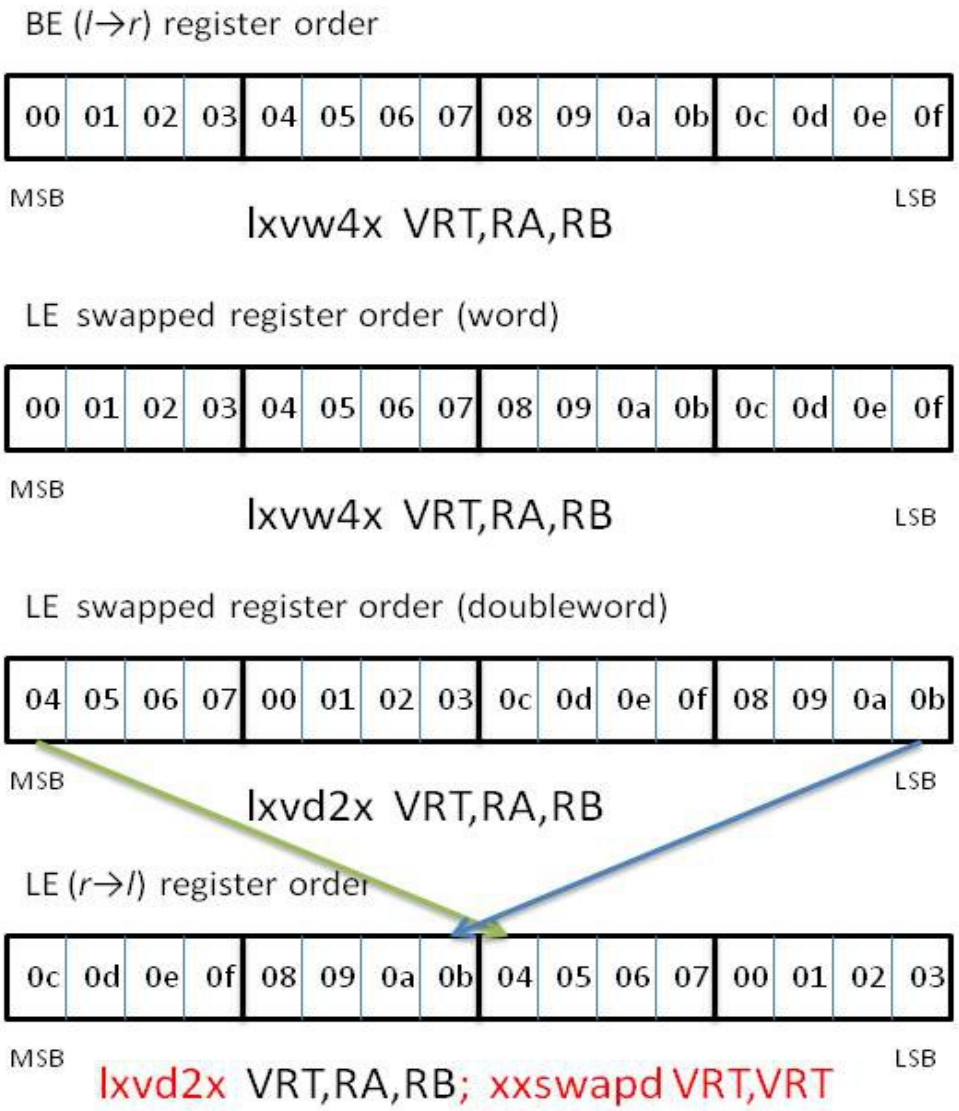


vnor VRC', VRC, VRC

vperm VRT, VRB, VRA, VRC'

Example: vec_xl for <4 x i32>





Outline

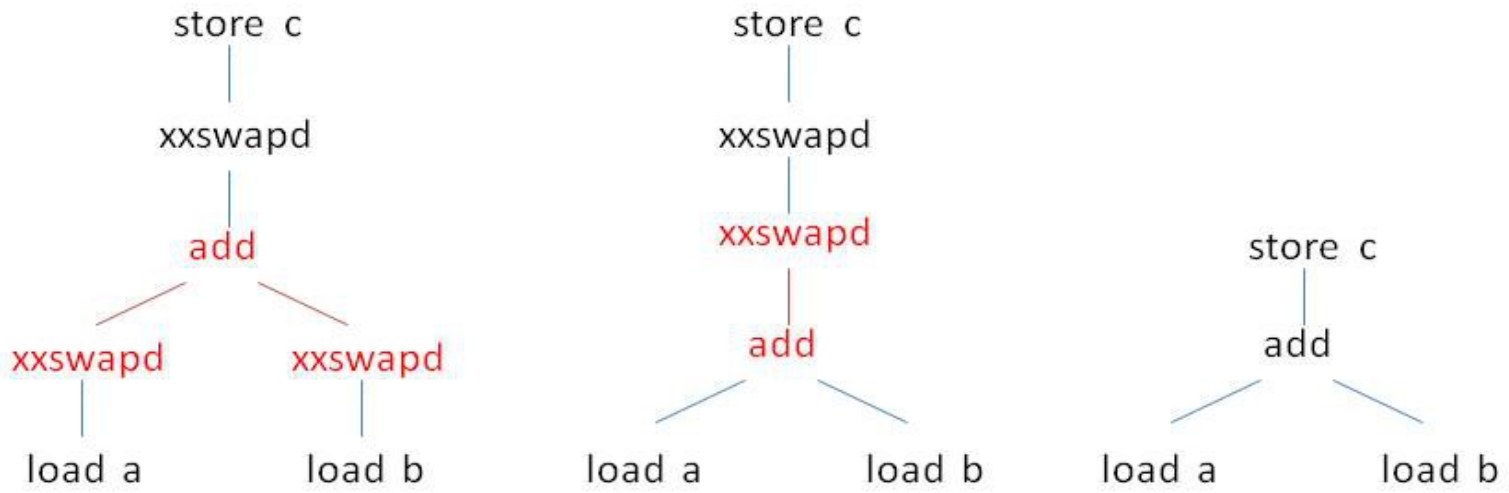
- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

Optimization – swap removal

- VMX load/store instructions may be preferred by compiler when 128-bit alignment guaranteed
  - Operate as expected for little endian
- But poor choice otherwise
  - Low-order 3 address bits are stripped
  - Two vector loads, formation of permute control vector, and a permute needed to emulate unaligned load
  - Only 32 vector registers
- VSX lxvd2x, lxvw4x, stxvd2x, stxvw4x provide good unaligned performance on POWER8, and 64 vector registers.
  - But now we have extra swap costs.
- If we can rid ourselves of the swaps, BE and LE performance are comparable.

Optimization – swap removal

- Observation: Most operations are pure-SIMD and don't care which lanes are used for computation, provided correct memory order is maintained.
  - Thus we look for computations where no swaps are needed.
- “Local” optimization (tree rewrites)



Optimization – swap removal

- Global optimization
  - Local optimizations work well on extended basic blocks.
  - But loop-carried dependencies, join points interfere
  - Auto-vectorized code is a common case that needs more.
- With du- and ud-chains available, simple to identify maximal vector computations with swaps at the boundaries
  - If all vector computations are lane-insensitive, remove swaps
  - Special handling for ops that are lane-sensitive
    - E.g., merge-high => merge-low with swapped inputs
  - Cost-modeling if special handling isn't “free”
  - Essentially linear time (union-find)

Outline

- History
- Little Endian on Power
- Endianness (bytes and elements)
- Programming models
- Example vector interfaces and implementations
- Optimization
- Implementation status (gcc, llvm)

Implementation status

- GCC
  - True-LE and BoL models are implemented (4.8+).
  - Global optimization of vector computations is implemented.
    - Most common special-handling cases are covered.
    - No cost modeling
    - Effective in optimizing most code with little effort
  - No local optimization (global is effective)
- LLVM
  - True-LE implemented for VMX instructions (3.5)
  - VSX not yet enabled for BE, let alone LE – in progress
  - No plans for BoL model at this time
  - Plan to implement True-LE for VSX with optimization

Acknowledgments and Links

- Acknowledgments
  - Kit Barton, David Edelsohn, Jinsong Ji, Ke Wen Lin, Joan McComb, Ian McIntosh, Steve Munroe, Hong Bo Peng, Julian Wang, Ulrich Weigand, Rafik Zurob
- Links
  - Power ISA 2.07:
    http://ibm.biz/powerisa
  - Power Architecture 64-Bit ELF V2 ABI Specification:
    http://ibm.biz/power-abi (free registration required)
  - AltiVec Technology Programming Interfaces Manual (1999):
    http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf

Questions?