Vector Lane Threading

S. Rivoire, R. Schultz, T. Okuda, C. Kozyrakis

Computer Systems Laboratory

Stanford University

Motivation

- □ Vector processors excel at **data-level parallelism (DLP)**

- □ What happens to program phases with little or no DLP?

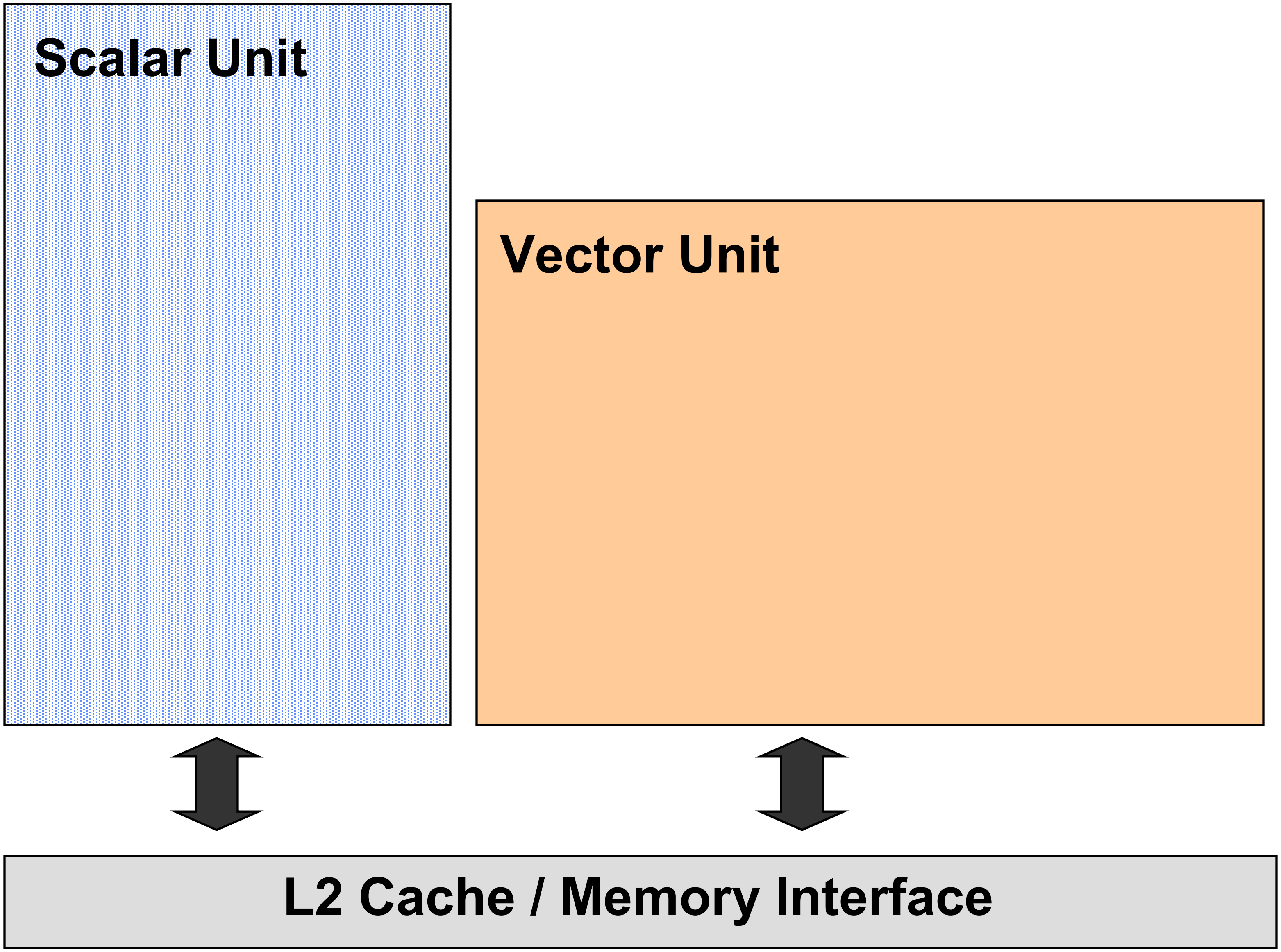
- □ **Vector Lane Threading (VLT)**
  - Leverage idle DLP resources to exploit **thread-level parallelism (TLP)**
  - 1.4-2.3x speedup on already optimized code
  - Small increase in system cost

- □ VLT increases the applicability of vector processors
  - Efficient for both regular & irregular parallelism

Outline

- ❑ Motivation
- ❑ Vector processor background
  - Generic vector microarchitecture
  - Vector processors and DLP
- ❑ Vector lane threading (VLT)
- ❑ VLT evaluation
- ❑ Conclusions

Vector Microarchitecture Overview

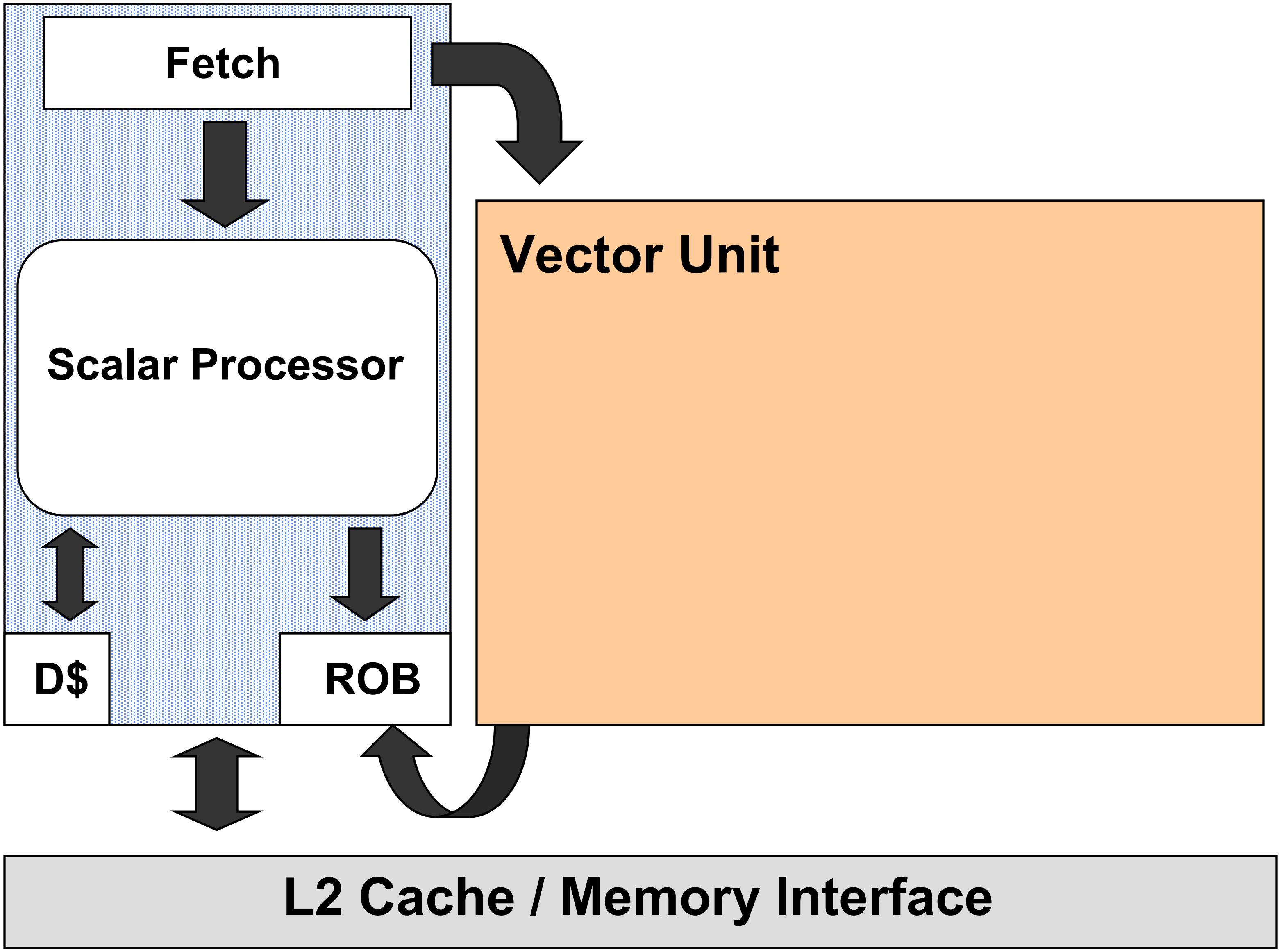


Scalar Unit

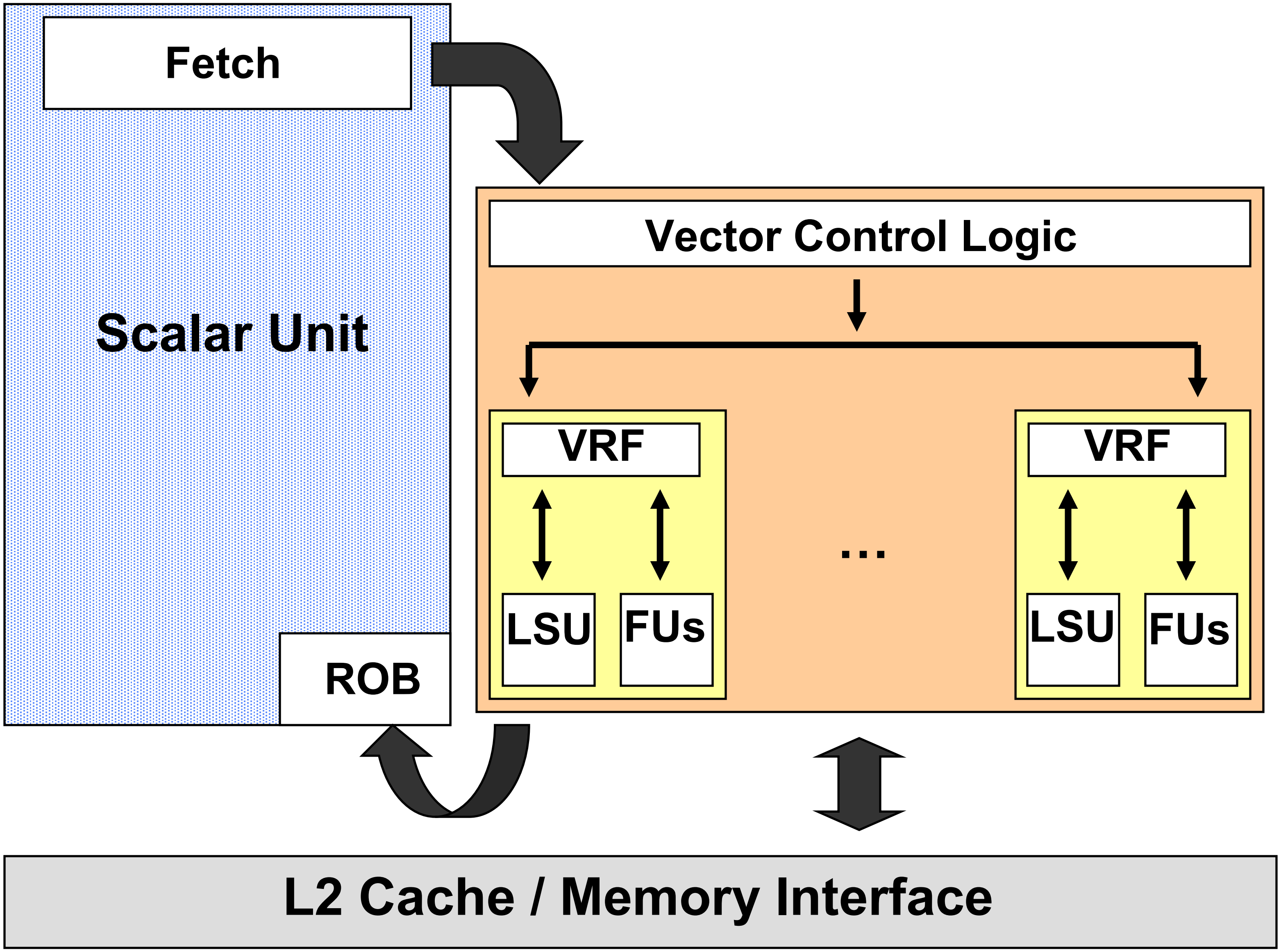
Vector Unit

L2 Cache / Memory Interface

Vector Microarchitecture Overview

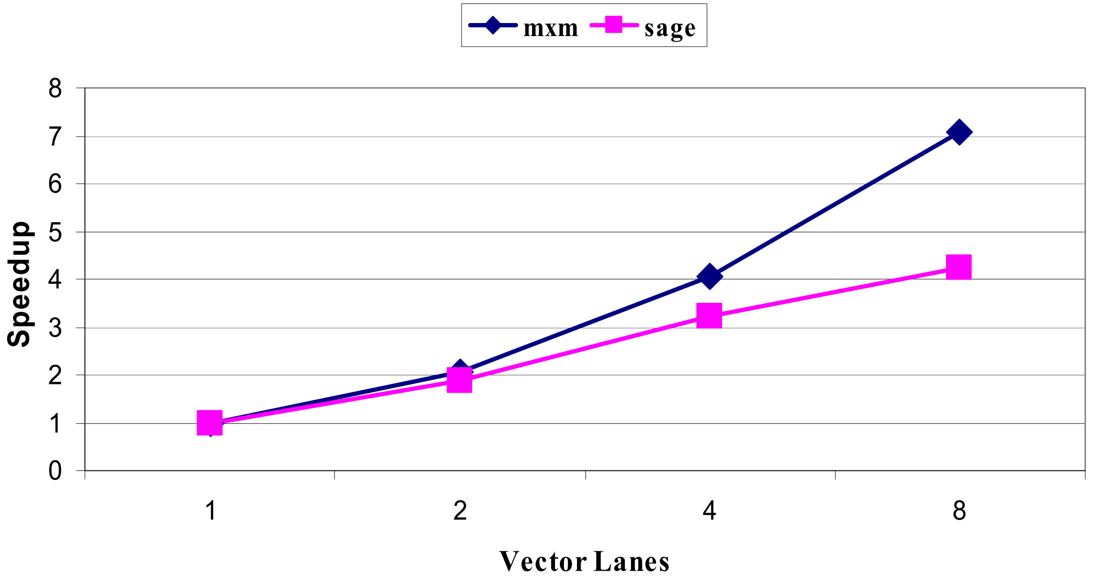


Vector Microarchitecture Overview



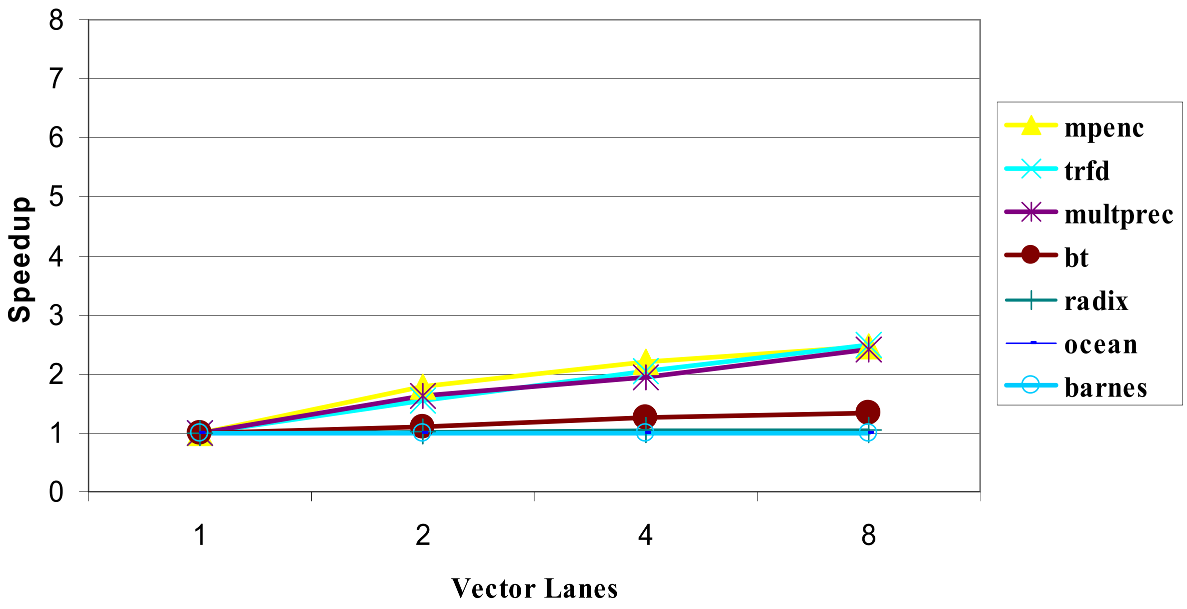
Vector Efficiency with High DLP

```
for i = 1 to N
  for j = 1 to N
    for k = 1 to N
      C[i,j] = C[i,j]+A[i,k]*C[k,j]
```



- ❑ Best case for vector processors: long vectors & regular memory patterns
- ❑ *Lanes* execute data-parallel operations very efficiently
  - Low instruction issue rate, simple control, compact code, power efficiency
- ❑ Simple model for scalable performance
  - Current vector processors have 4 to 16 lanes

Vector Efficiency with Low DLP



- ❑ Low DLP ⇒ underutilized lanes & memory ports
  - Short vectors
  - No vectors
  - Vector length vs. stride in nested loops
- ❑ Can we improve efficiency for low-DLP cases?

Outline

❑ Motivation

❑ Vector processor background

❑ Vector lane threading (VLT)

- Overview
- Possible configurations
- Implementation

❑ VLT Evaluation

❑ Conclusions

VLT Basics

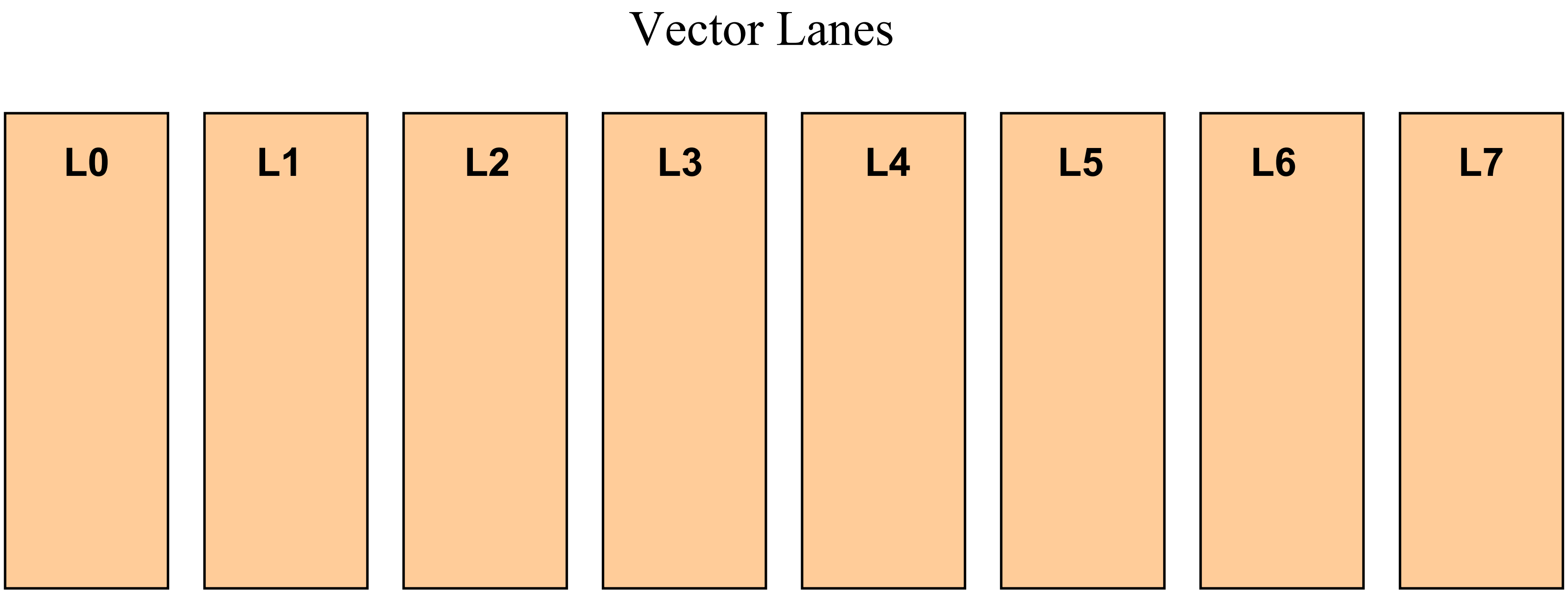
- ❑ Idea: use idle lanes to exploit thread-level parallelism (TLP)

- ❑ Sources of TLP
  - Outer loops in loop nests
  - Non-vectorizable loops
  - Task-level parallelism

- ❑ VLT benefits
  - Does not harm high-DLP performance
  - Higher utilization for DLP resources
  - Vector unit can be shared between multiple threads
    - From SMT or CMP scalar processors

VLT Configurations: Single Vector Thread

- Original configuration for long vector lengths (high DLP)
  - 1 thread running vector instructions on 8 lanes

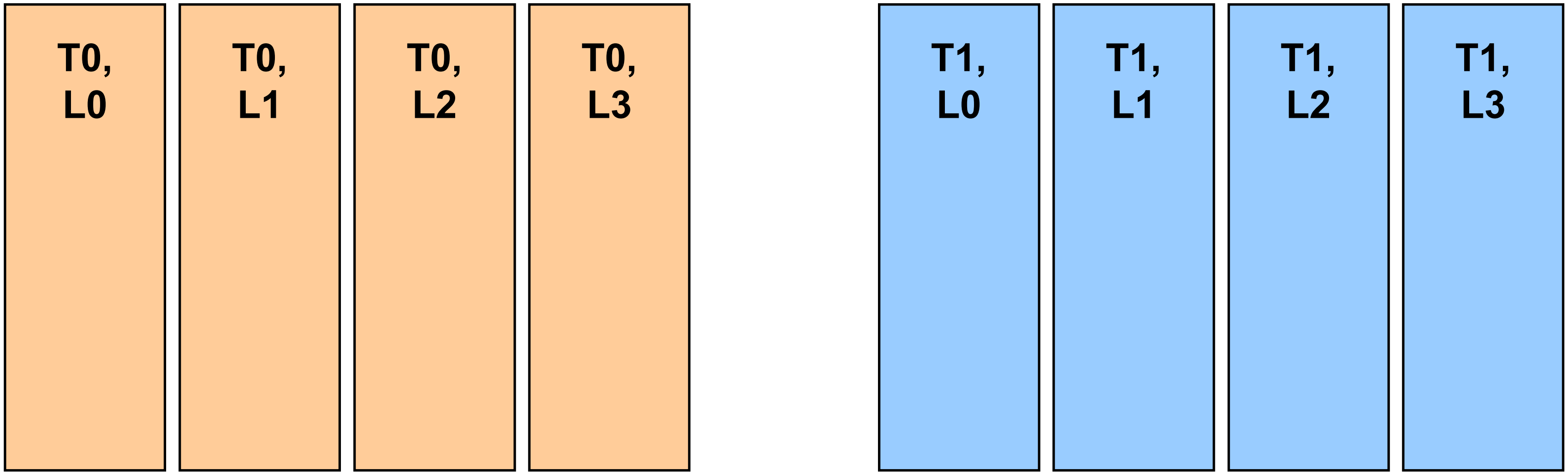


VLT Configurations: 2 Vector Threads

❑ Medium vector length configuration

- Two threads, each running vector instructions on 4 lanes
- Threads may be controlled by SMT or CMP scalar processor (more later)

Vector Lanes



T0, L0

T0, L1

T0, L2

T0, L3

T1, L0

T1, L1

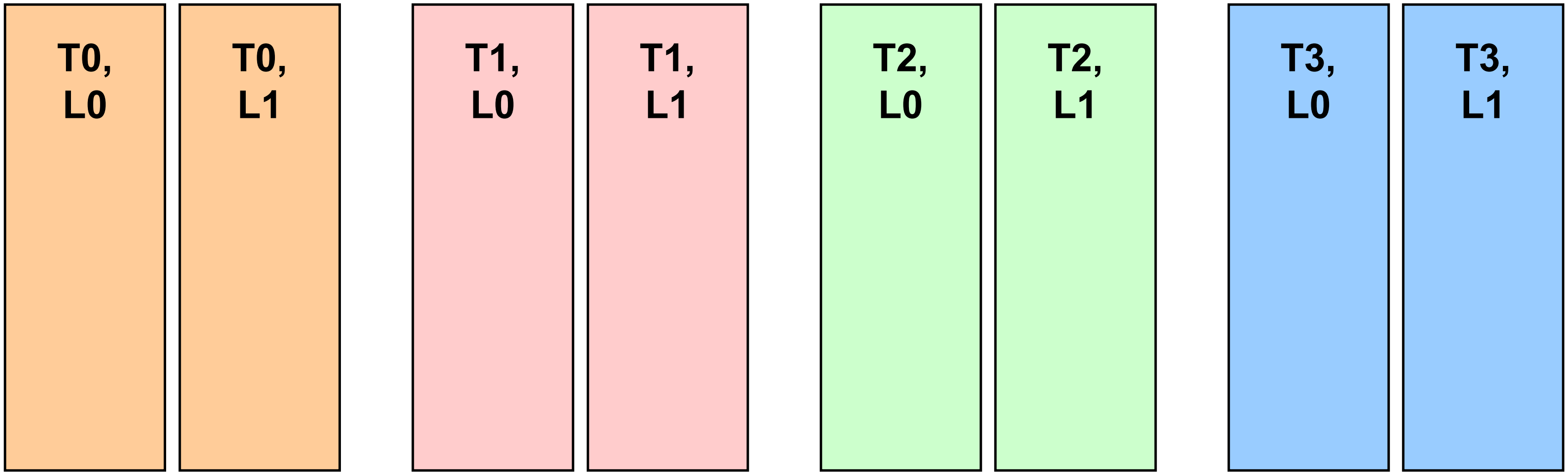
T1, L2

T1, L3

VLT Configurations: 4 Vector Threads

- Short vector length configuration
  - Four threads, each running vector instructions on 2 lanes
  - Threads may be controlled by SMT or CMP scalar processor (more later)

Vector Lanes



T0, L0

T0, L1

T1, L0

T1, L1

T2, L0

T2, L1

T3, L0

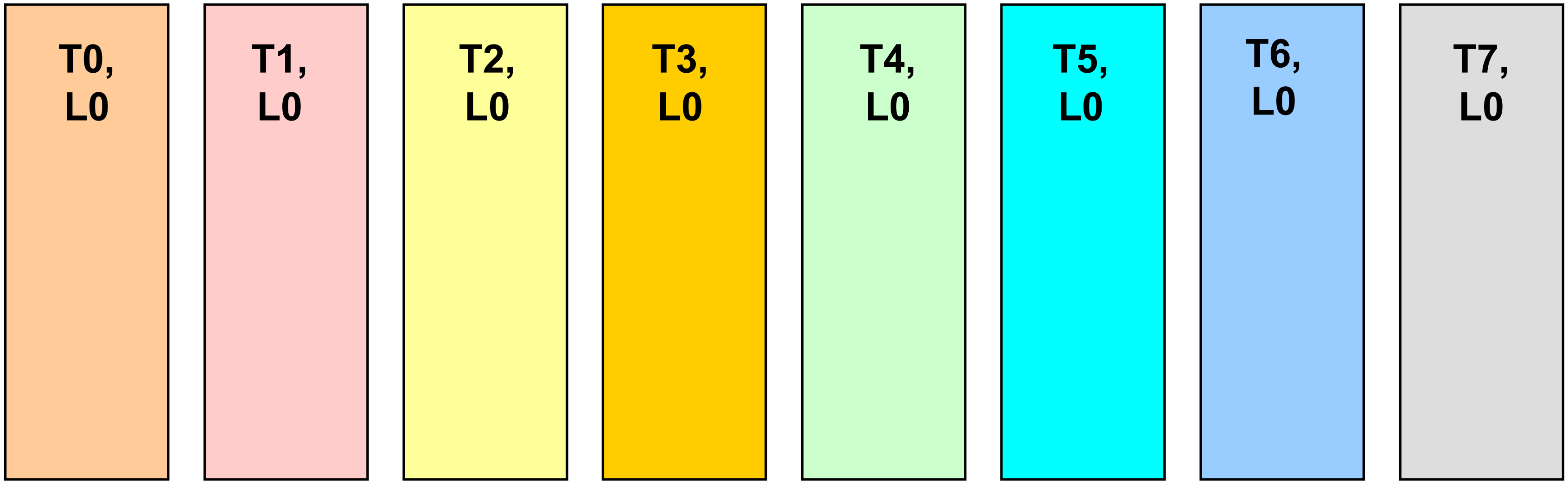
T3, L1

VLT Configurations: Pure Scalar Threads

❑ Scalar (no-vector) configuration

- Eight threads, each running scalar instructions on 1 lane
- Each lane operates as a simple processor

Vector Lanes



T0, L0

T1, L0

T2, L0

T3, L0

T4, L0

T5, L0

T6, L0

T7, L0

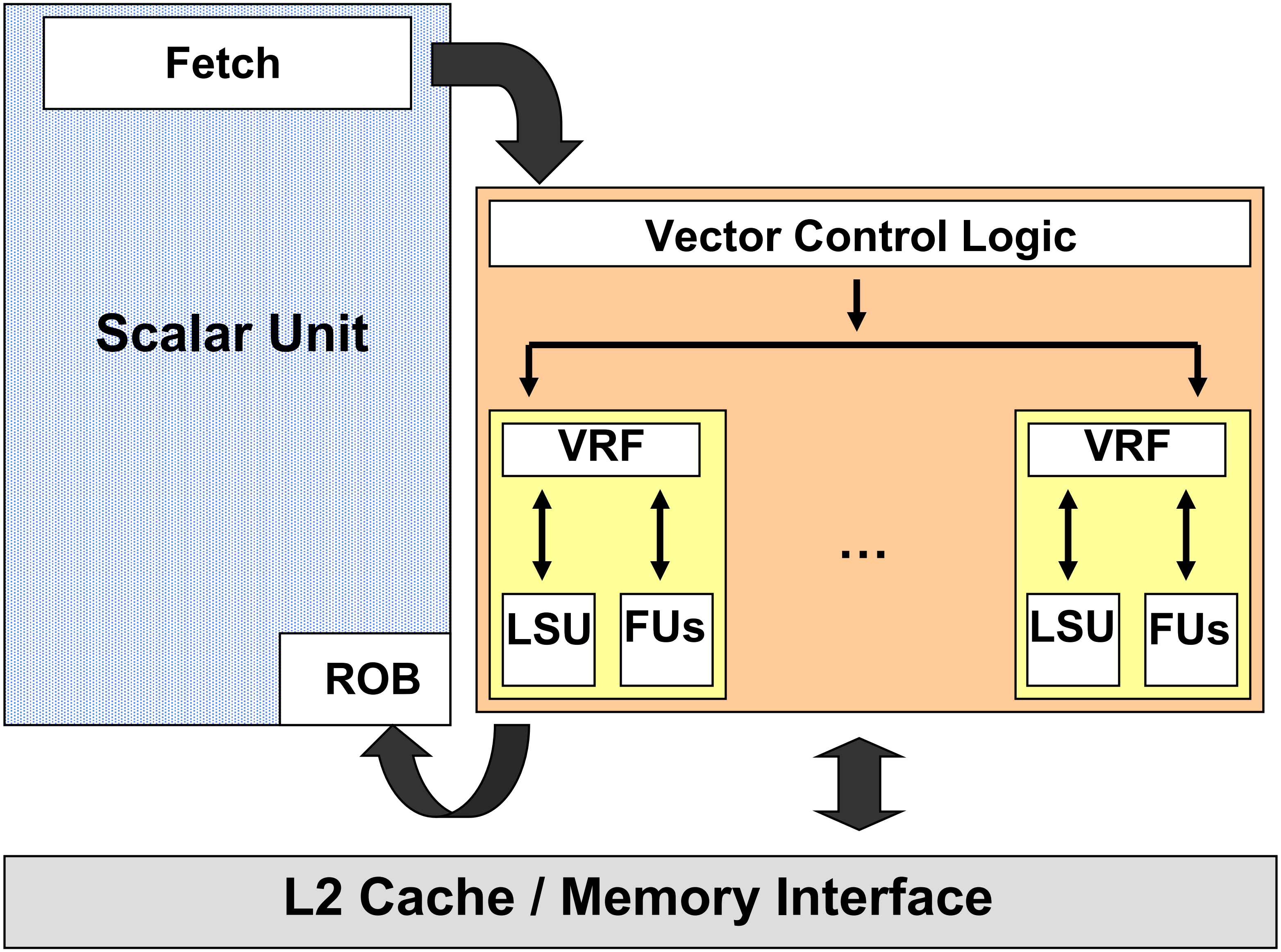
VLT vs. SMT

- VLT: exploit TLP on idle DLP resources
  - Different threads on different vector lanes
  - Each thread uses all functional units within a lane

- SMT: exploit TLP on idle ILP resources
  - Different threads on different functional units
  - In a vector processor, each thread uses all lanes

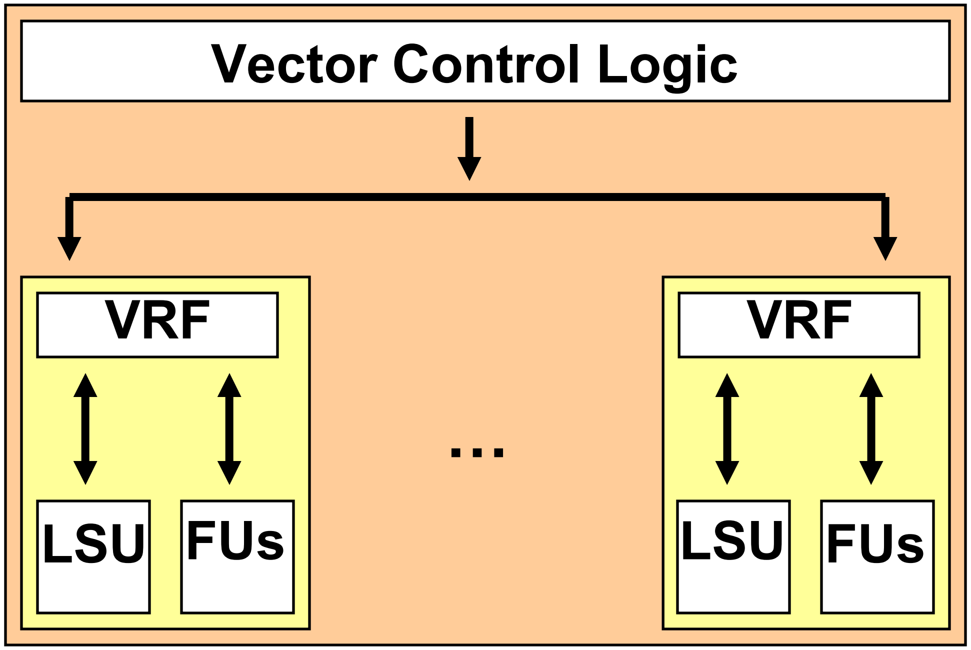
- Notes
  - VLT and SMT are orthogonal & can be combined
  - VLT is more important for a vector processor
    - Several apps run efficiently on a 16-lane vector processor
    - How many apps run efficiently on a 16-way issue processor?

VLT Implementation



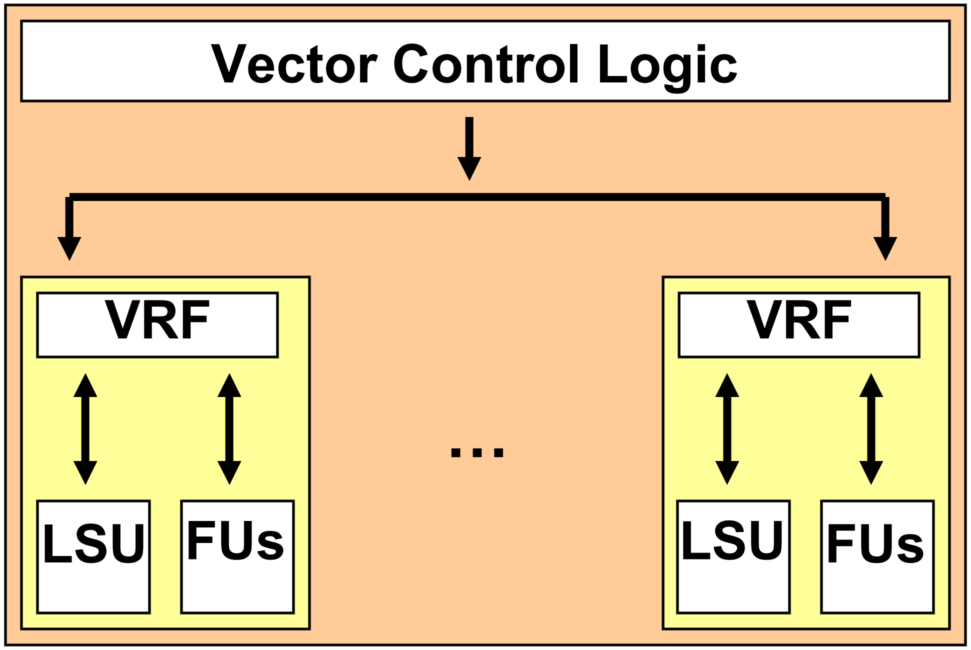
VLT Implementation: Execution Resources

- ❑ Functional units ⇒ already available
- ❑ Vector registers for additional threads ⇒ already available
  - VRF slice in each lane can store the register elements for each thread
  - Note that each thread uses shorter vectors
- ❑ Memory ports ⇒ already available
  - Necessary to support indexed and strided patterns even with long vectors

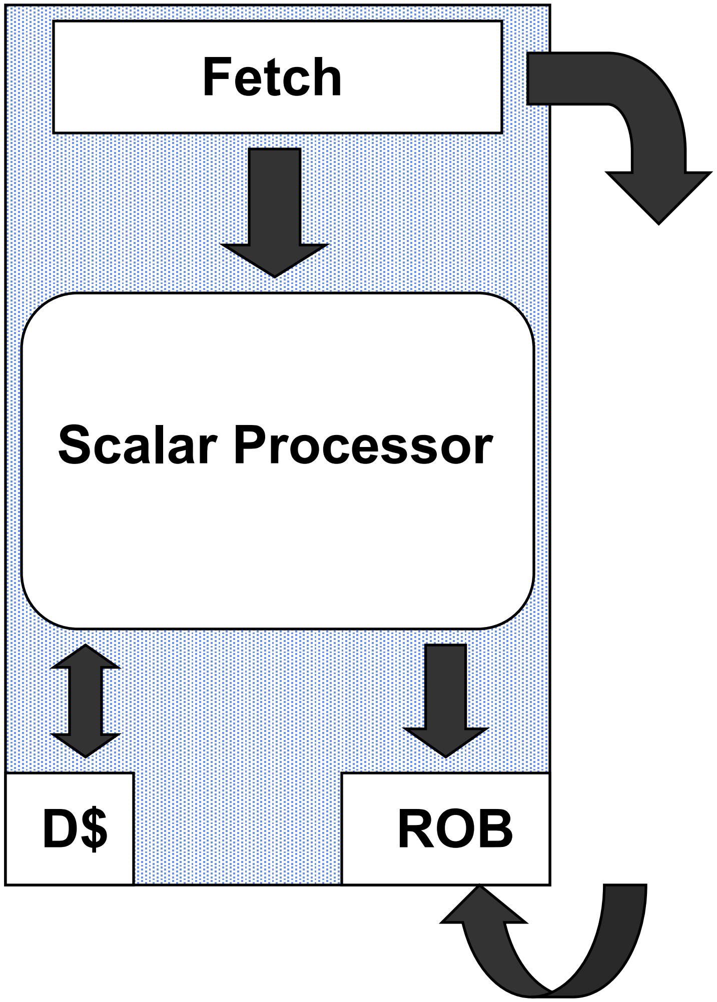


VLT Implementation: Vector Instr. Bandwidth

- Vector instruction issue bandwidth
  - Must issue vector instructions separately for each thread
  - *Multiplex* single vector control block, or *replicate* the block
  - Multiplexing is sufficient as each vector instruction takes multiple cycles
- Instruction set
  - Minor addition to specify configuration (number of threads)



VLT Implementation: Scalar Instr. Bandwidth



- ❑ Must process scalar instructions for multiple vector threads

- ❑ Design alternatives
  - Attach vector unit to SMT processor
  - Attach vector unit to CMP processor
    - Multiple cores share one vector unit
  - Combination of the above (CMT)
  - Heterogeneous CMP
    - One complex & multiple simpler cores share a vector unit

- ❑ Trade-off
  - Cost vs. utilization of vector unit

VLT Implementation: Scalar Threads on Vector Unit

- ❑ Challenge: each lane requires 1-2 instructions per clock cycle
  - Much higher instruction bandwidth than with vector threads
  - No point in using multiple scalar cores to feed the lanes

- ❑ Design approach
  - Introduce a small instruction cache in each lane
    - Single-ported, 1 to 4 Kbytes is sufficient
    - Feeds the functional units with instructions in tight loops
  - Cache misses in lanes handled by scalar core
    - Scalar core instruction cache acts as a L2 instruction cache
    - Low miss penalty

Outline

- ❑ Motivation
- ❑ Vector processor background
- ❑ Vector lane threading (VLT)
- ❑ VLT Evaluation
  - Methodology
  - Vector thread results
  - Scalar thread results
- ❑ Conclusions

Methodology

- ❑ Simulated processor
  - Based on Cray X1 ISA
  - Scalar unit: 4-way OOO superscalar
  - Vector registers: 32 vector registers, max. vector length of 64
  - Vector execution resources: 8 lanes, 3 functional units per lane
- ❑ Software environment
  - All code compiled with production Cray C and Fortran compilers
  - Highest level of vector optimizations enabled
    - All speedups reported are in addition to vectorization
  - Used ANL macros for multithreading
    - Could also use OpenMP or other mulithreading approaches
- ❑ Further details in the paper

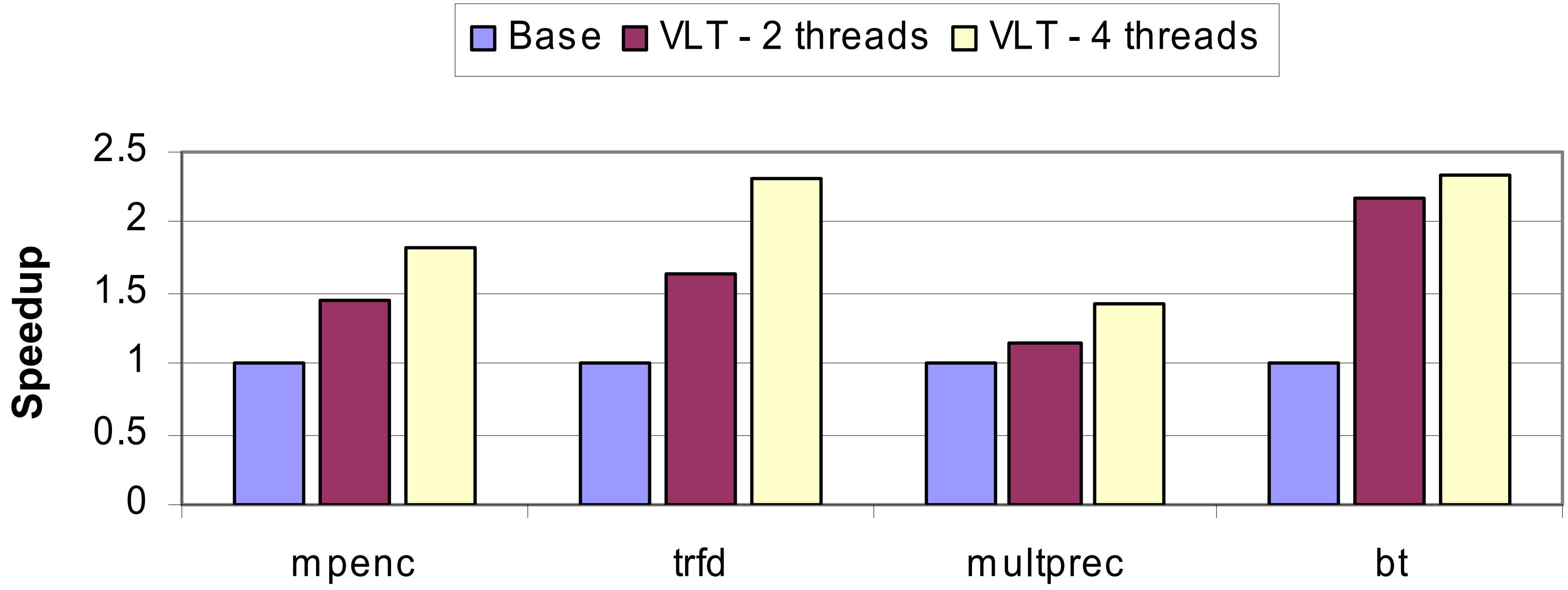
Benchmarks

- Nine benchmarks, three categories
  - High DLP (long vectors), medium DLP (short vectors), no DLP

- Examples
  - High DLP: matrix multiply
    - 96% vectorized, average vector length of 64 (maximum)
  - Medium DLP: trfd (two electron integral transformation)
    - 76% vectorized, average vector length of 11.2
  - No DLP: ocean (eddy currents in ocean basin)
    - 0% vectorized

- Note
  - Benchmarks include some sequential portions with no DLP, no TLP

Vector Thread Evaluation



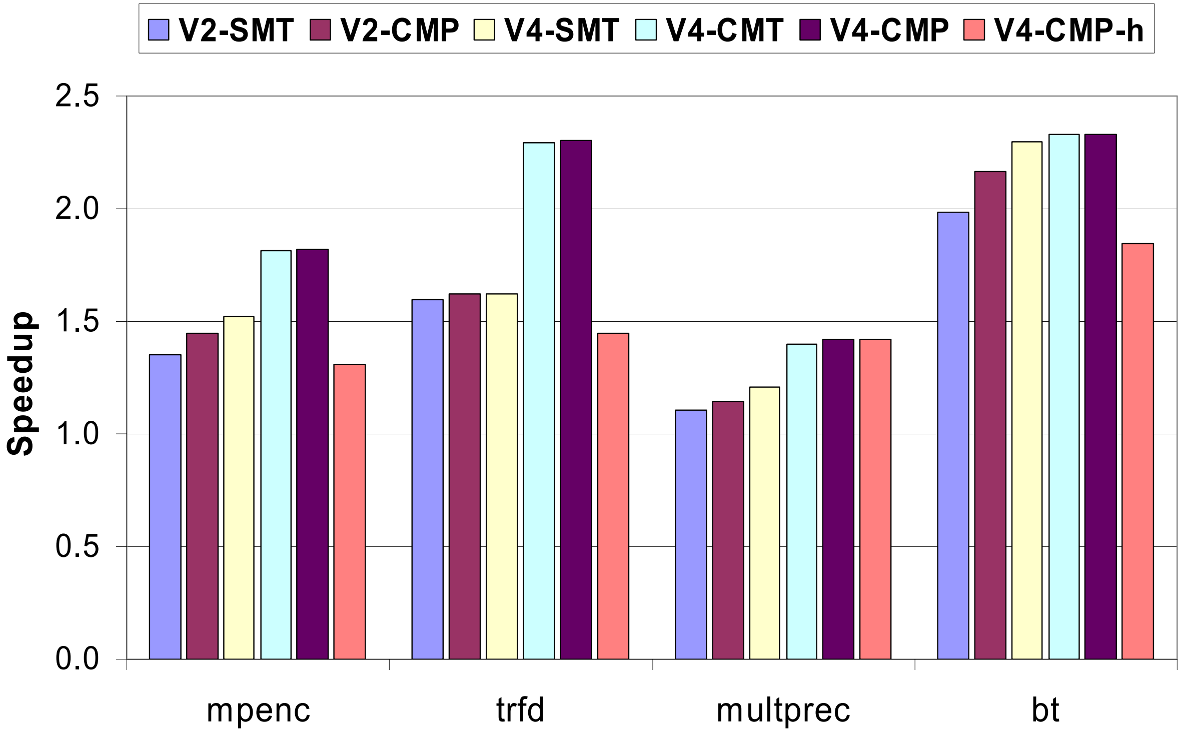
- ❑ Results for medium-DLP benchmarks on best scalar-core configuration
- ❑ Up to 2.3x performance improvement
  - On top of vectorization for these applications
- ❑ Limitations
  - Saturation of vector resources
  - Cache effects, thread overhead, purely sequential portions

VLT Cost Evaluation

- ❑ Default configuration: SU (4-way OOO) + VU

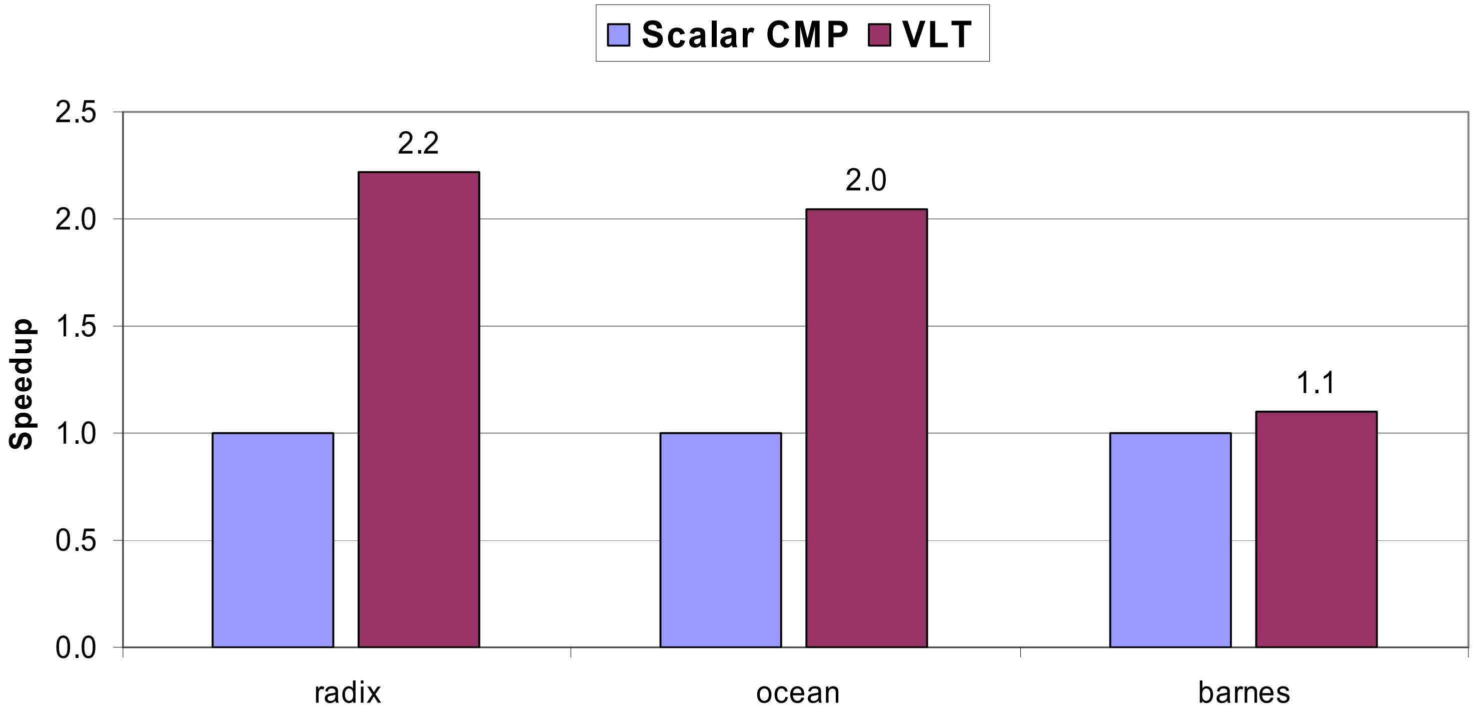
- ❑ SMT scalar unit
  - 0.8% area increase for 2 VLT threads
  - 1.3% area increase for 4 VLT threads
- ❑ CMP scalar unit (4-way OOO cores)
  - 12.3% area increase for 2 VLT threads
  - 26.9% area increase for 4 VLT threads
- ❑ CMT scalar unit (4-way OOO cores, 2 threads/core)
  - 13.8% area increase for 4 VLT threads
- ❑ Heterogeneous CMP scalar unit (CMP-h)
  - 3.4% area increase for 2 VLT threads
  - 10.1% area increase for 4 VLT threads

VLT Design Space Evaluation



- V4-CMT equal performance to V4-CMP
  - At lower area increase (13.8% vs. 26.9%)
  - Two 4-way OOO processors can saturate an 8-lane vector unit
- V4-SMT outperforms V4-CMP-h
  - V4-CMP-h configuration suffers thread imbalance

Scalar Thread Evaluation



- ❑ 8-lane VLT operates like a 8-way CMP with very simple cores
  - No out-of-order execution, wide issues, branch prediction, etc
- ❑ Up to 2x compared to CMP (4-way cores with 2 threads/core)
  - But performance may vary due to sequential code performance

Conclusions

- ❑ Vector Lane Threading (VLT)
  - Turn underutilized DLP resources (lanes) into TLP resources
  - Multiple scalar processors share a vector unit
  - Vector unit used as a CMP with very simple cores
- ❑ Results
  - Up to 2.3x performance for applications with short vectors
  - Up to 2.0x performance for applications with no vectors
  - Cost-effective implementation alternatives
- ❑ Overall, VLT improves the applicability of vector processors
  - Good efficiency with both high DLP and low DLP

Acknowledgments

❑ Cray

- X1 compiler access
- Research funding through DARPA HPCS

❑ Stanford Graduate Fellowship

❑ National Science Foundation Fellowships

❑ Stanford Computer Forum