
# Remote Store Programming

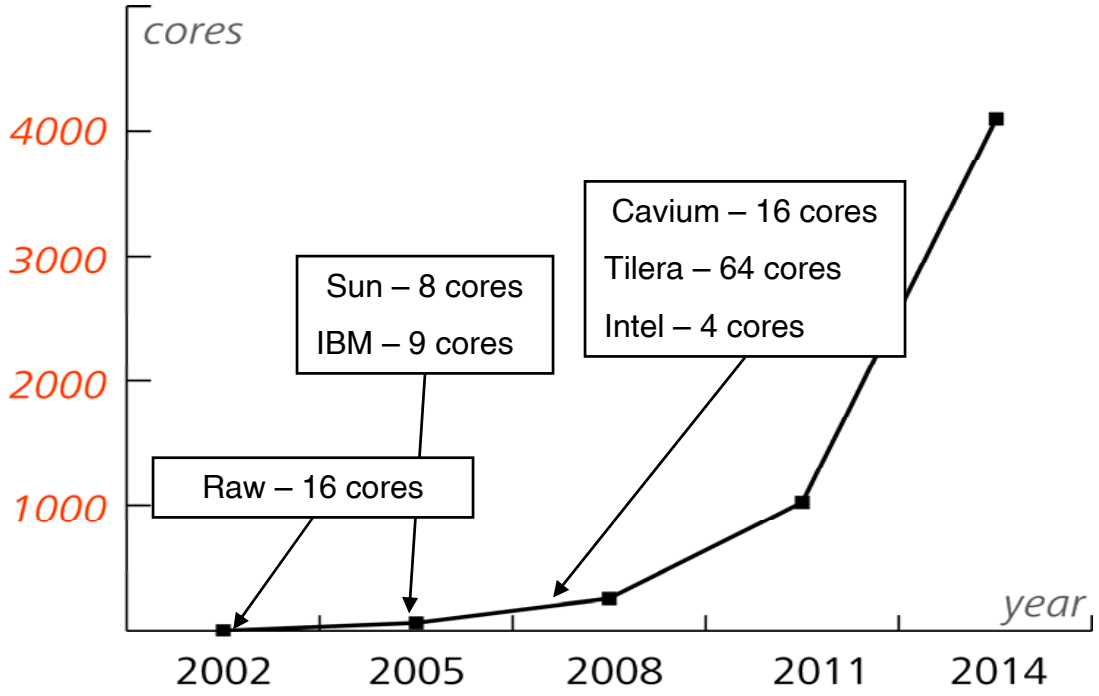
Henry Hoffmann  David Wentzlaff  Anant Agarwal

High Performance Embedded Computing Workshop
September 2009

# Remote Store Programming Outline

- Introduction/Motivation
- Key Features of RSP
  - Usability
  - Performance
- Evaluation
  - Methodology
  - Results
- Conclusion

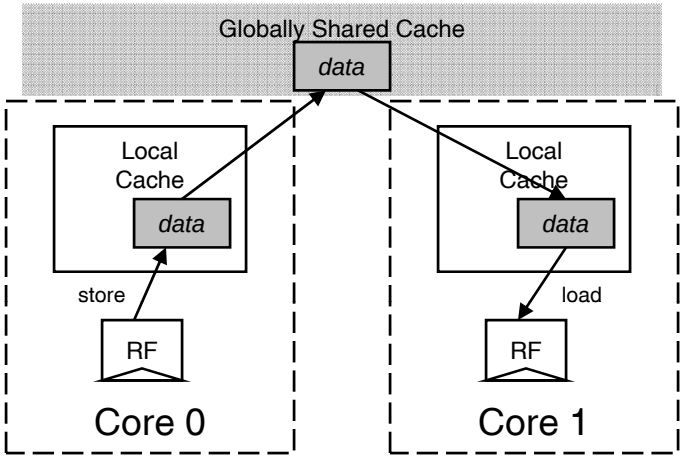
# Multicore requires innovation to balance usability and performance



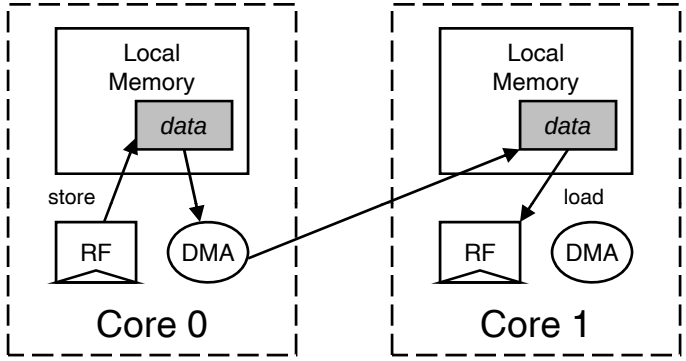

- Parallel programming is becoming ubiquitous
  - Parallel programming is no longer the domain of select experts
  - Balancing ease-of-use and performance is more important than ever

# Existing programming models do no combine usability and performance

Threads with cache coherence (CC)



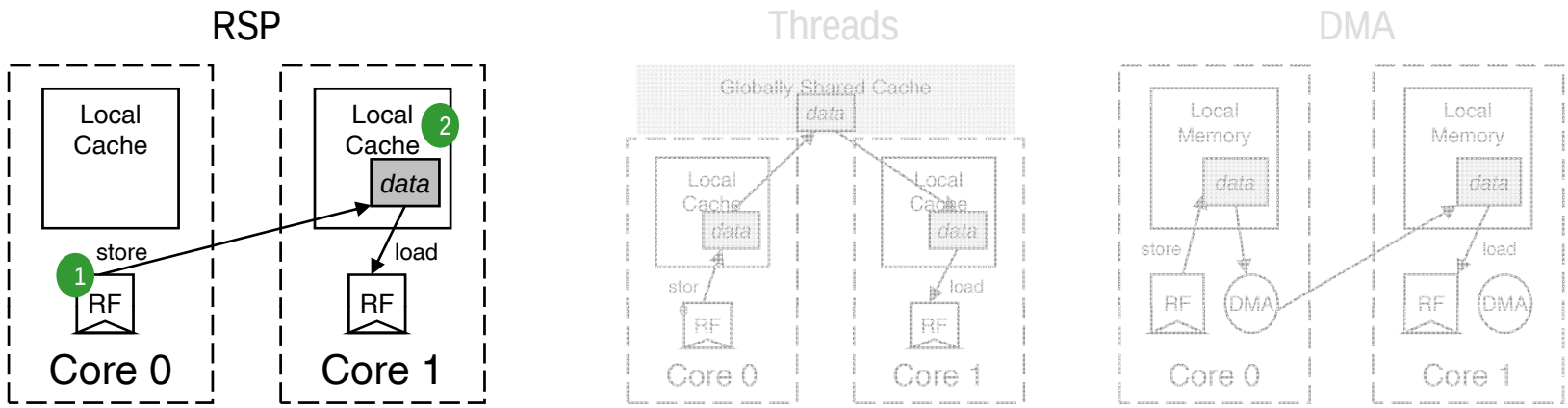

Direct Memory Access (DMA)




| | | | | |
|---|---|---|---|---|
| Usability | ✓ | • Familiar loads and stores<br>• Fine-grained comm. is easy | ✗ | • Requires additional software API<br>• Hard to schedule DMA transactions |
| Performance | ✗ | • No control over locality | ✓ | • Programmer has complete control over locality |

Remote Store Programming (RSP) can combine usability with performance

# RSP combines the usability of Threads with the performance of DMA




| | | | Implications |
|---|---|---|---|
| 1 Usability | ✓ | • Familiar loads and stores<br>• Fine-grained<br>• One-sided | Can develop high performance programs in a shorter amount of time |
| 2 Performance | ✓ | • Software controls locality | Always access physically close memory and minimize load latency |



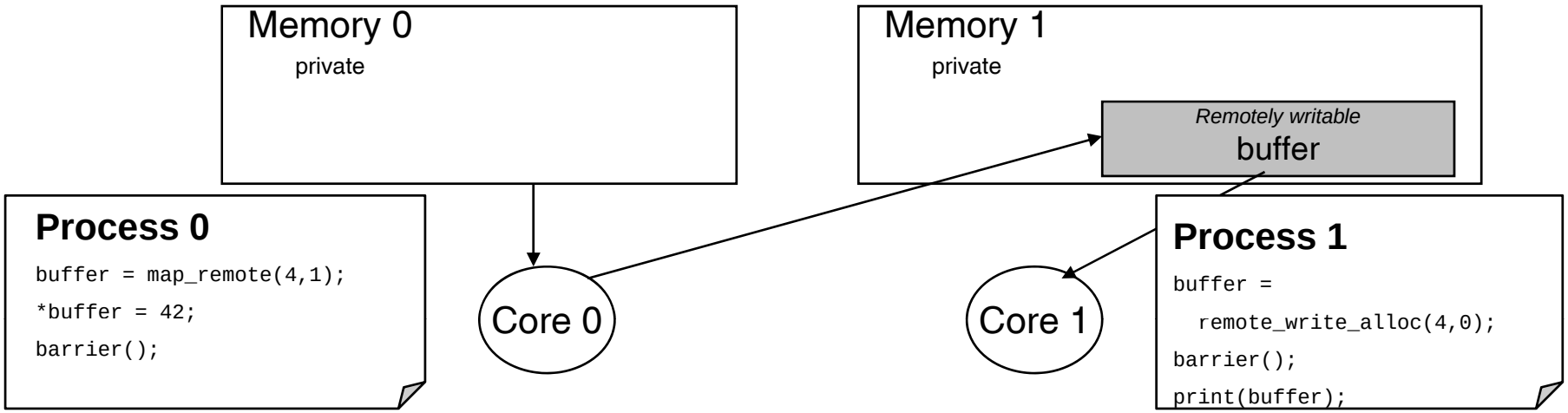
TILERA EMBEDDING MULTICORE

# Remote Store Programming Outline

- Introduction/Motivation
- Key Features of RSP
  - Usability
  - Performance
- Evaluation
  - Methodology
  - Results
- Conclusion

# RSP captures the usability of threads and shared memory

1 Usability
2 Performance




- Process Model
  - Each process has private memory by default
  - A process can grant write access to remote processes
- Communication
  - Processes communicate by storing to remotely writable memory
- Synchronization
  - Supports test-and-set, compare-and-swap, etc.
  - We assumer higher level primitives like barrier



TILERA EMBEDDING MULTICORE

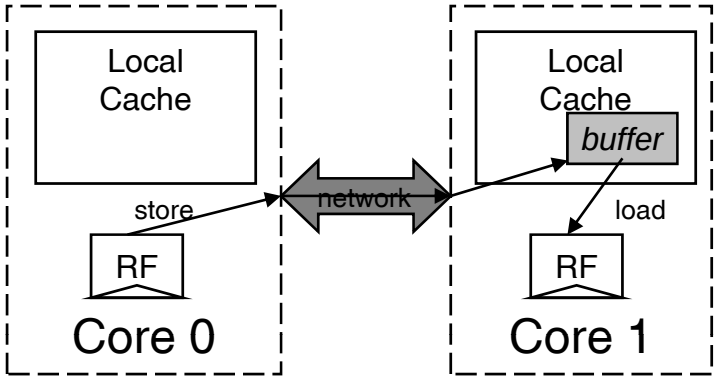
# RSP emphasizes locality for performance on large scale multicores

1 Usability

2 Performance

Core 0 is a producer:

Data transferred from Core 0's registers to Core 1's cache




Core 1 is a consumer:

All loads guaranteed to be local and minimum latency

| Step | Description |
|---|---|
| 1 | Core 1 gives Core 0 write permission for *buffer* |
| 2 | Core 0 receives write permission from Core 1 for *buffer* |
| 3 | Core 0's store instructions target remote memory on consumer (*buffer*) |
| 4 | Core 1's standard load instructions are used to read data from cache |

**Performance benefit**

**Locality**
- Cores access close memories
- Load latency is minimized

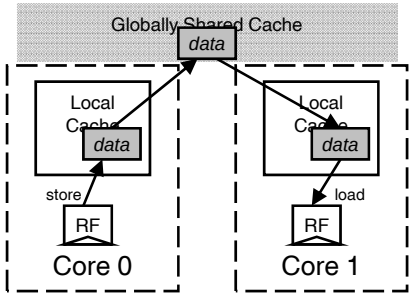
**Fine Grain Comm.**
- Complete overlap of communication and computation

# 2D FFT example illustrates performance and usability

1 Usability

2 Performance

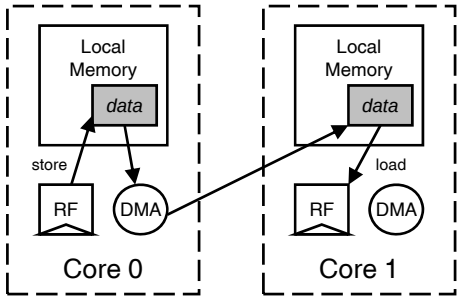
## Threads




```
Matrix A, C;
shared Matrix B;
pid = my_id();
init(A);
row_fft(B,A, pid);
barrier();
col_fft(C,B,pid);
```

Easiest to write ✓
Fine-grain ✓
No locality ✗

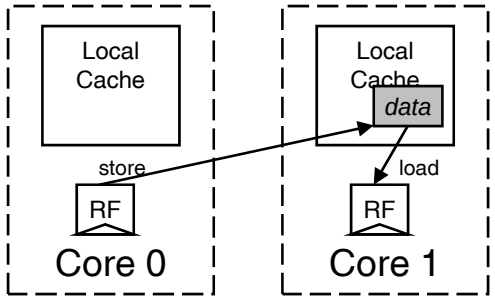
## DMA




```
Matrix A, B, B2, C;
pid = my_id();
init(A);
row_fft(B,A, pid);
DMA_move(B2,B);
col_fft(C,B2,pid);
```

Hard to write ✓
Coarse-grain ✓
High locality ✓

## RSP




```
Matrix A, C;
Matrix B =
   alloc_rsp();
pid = my_id();
init(A);
row_fft(B,A, pid);
barrier();
col_fft(C,B,pid);
```

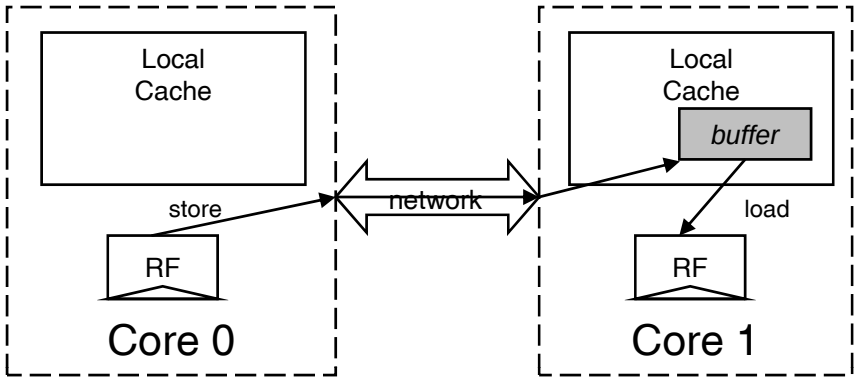
Easy to write ✓
Fine-grain ✓
High locality ✓

For more detail see: Hoffmann, Wentzlaff, Agarwal. Remote Store Programming: Mechanisms and Performance. Technical Report MIT-CSAIL-TR-2009-017. May, 2009



TILERA EMBEDDING MULTICORE

# RSP requires incremental hardware support

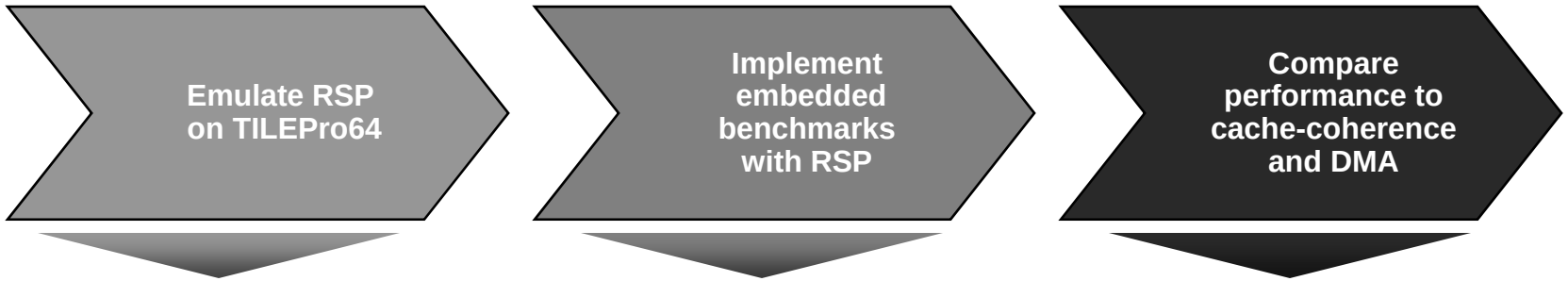



- RSP requires incremental additional hardware
  - In processor supporting cache coherent shared memory
    - Additional memory allocation mode for remotely writable memory
    - Do not update local cache when writing to remote memory
  - In processor without cache coherent shared memory
    - Memory allocation for remotely writable memory
    - Write misses on remote cores forward miss to allocating core

# Remote Store Programming Outline

- Introduction/Motivation
- Key Differentiators of RSP
  - Usability
  - Performance
- Evaluation
  - Methodology
  - Results
- Conclusion

# RSP, Threads, and DMA are compared on the TILEPro64 processor

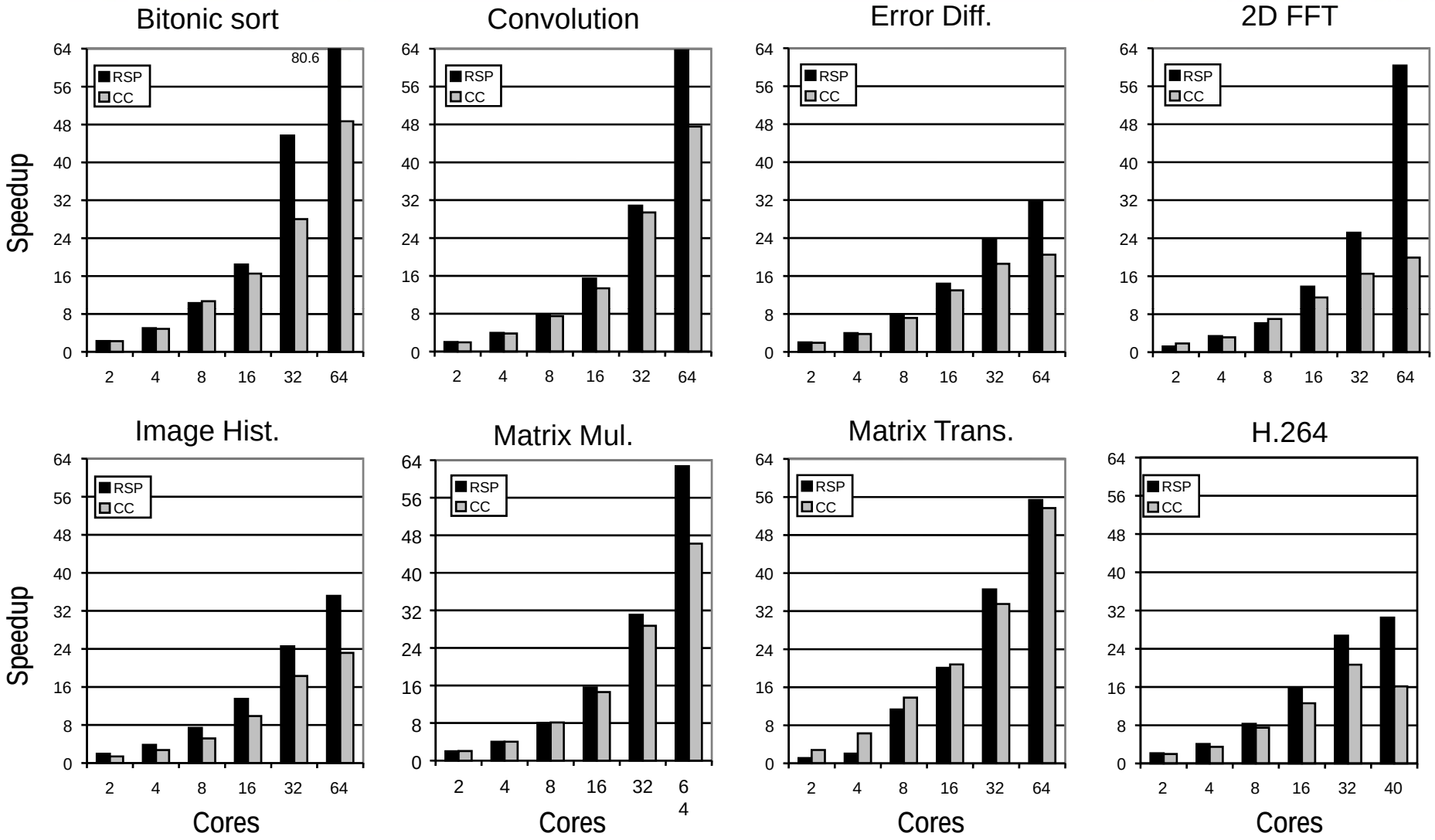



- TILEPro64 supports Cache-coherence and DMA
- Additional memory allocation modes let us emulate RSP

- Matrix Transpose
- Two-dimensional FFT
- Matrix Multiply
- Bitonic Sort
- Convolution
- Floyd-Steinberg Error Diffusion
- H.264 Encode
- Histogram

- Compare speedup and load latency

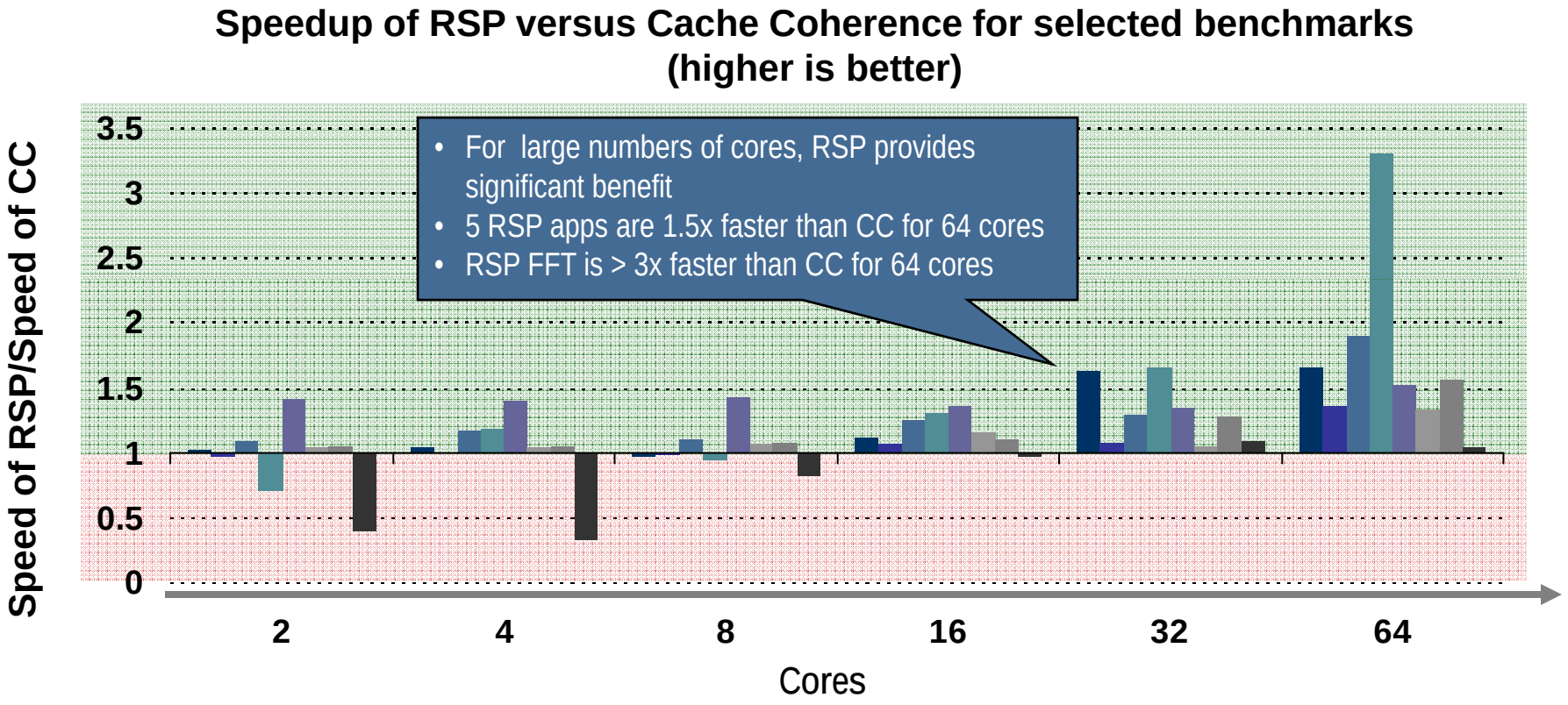
# Speedups of RSP and cache-coherent benchmarks



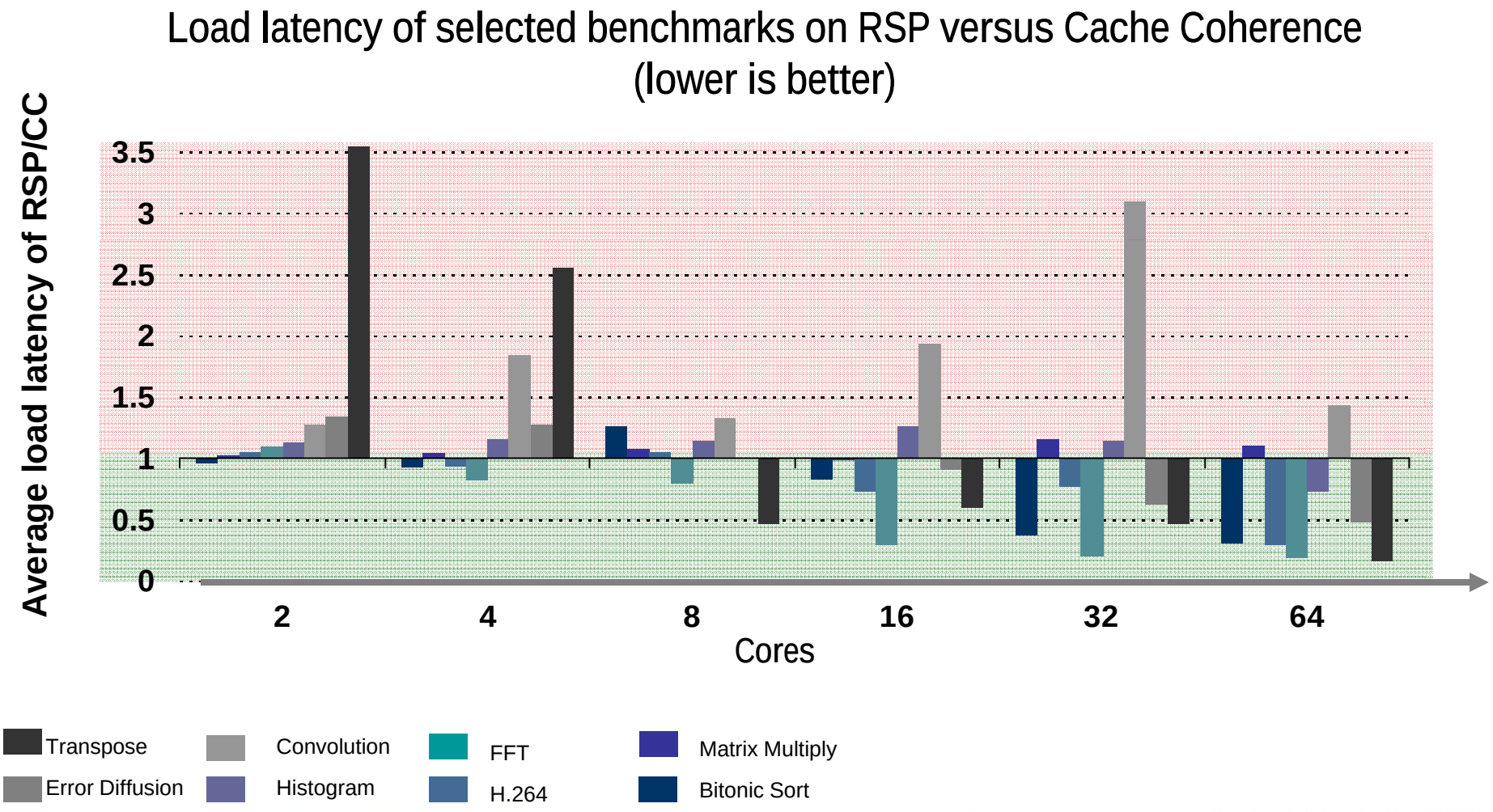



TILERA™ EMBEDDING MULTICORE

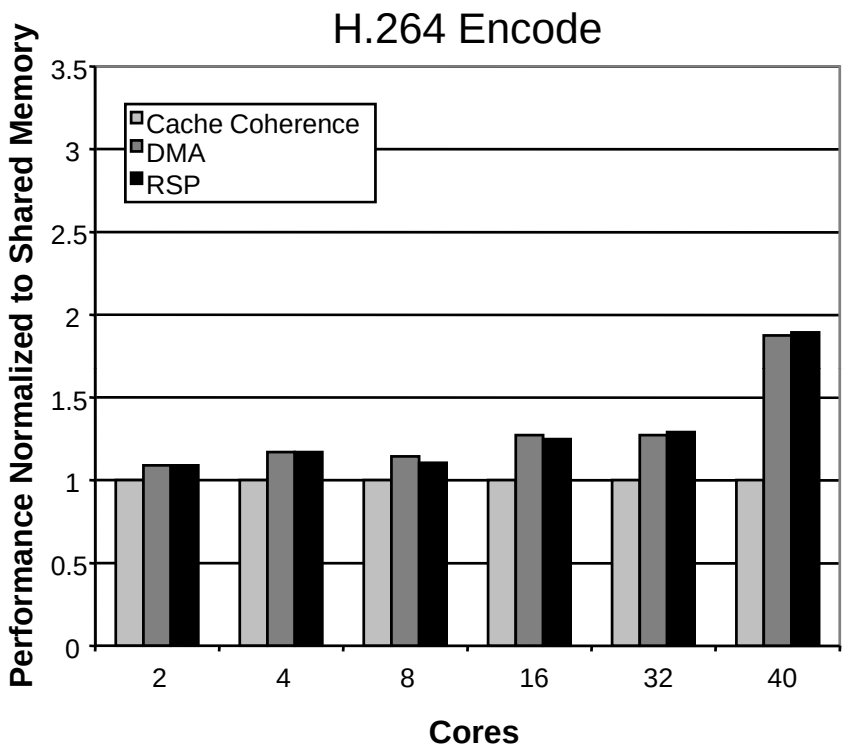
# RSP is outperforms threading with cache coherence for large numbers of cores

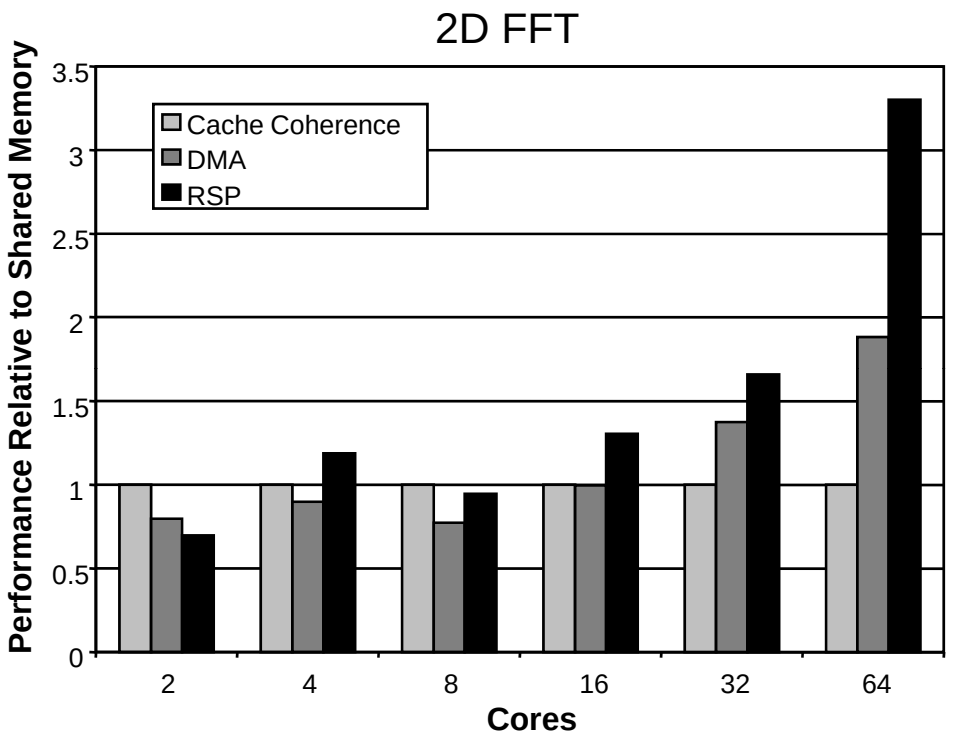


# RSP's emphasis on locality results in low load latency



# Comparison of RSP, Threading, and DMA for two applications




- RSP is faster than Threading and DMA due to fine emphasis on locality and fine-grain communication

# Remote Store Programming Outline

- Introduction/Motivation
- Key Differentiators of RSP
  - Usability
  - Performance
  - Hardware Requirements
- Evaluation
  - Methodology
  - Results
- Conclusion

# Conclusion

- Talk presents Remote Store Programming
  - A programming model for multicore
  - Uses familiar communication mechanisms
  - Achieves high-performance through locality and fine-grain communication
  - Requires incremental hardware support

- Conclusion:
  - Threads and shared memory good performance and easy to use
  - For large numbers of cores RSP can outperform threading because of greater locality
  - RSP is easier to use than DMA and slightly harder than Threads

- Anticipated use for RSP:
  - Can supplement threads and cache-coherent shared memory on multicore
    - Most code uses standard threading and shared memory techniques
    - Performance critical sections of code or applications use RSP for additional performance
    - Gentle Slope to programming multicore