# Jacobi Red-Black
# CUDA vs Xeon vs Xeon Phi

Ed Karrels

# Performance comparison

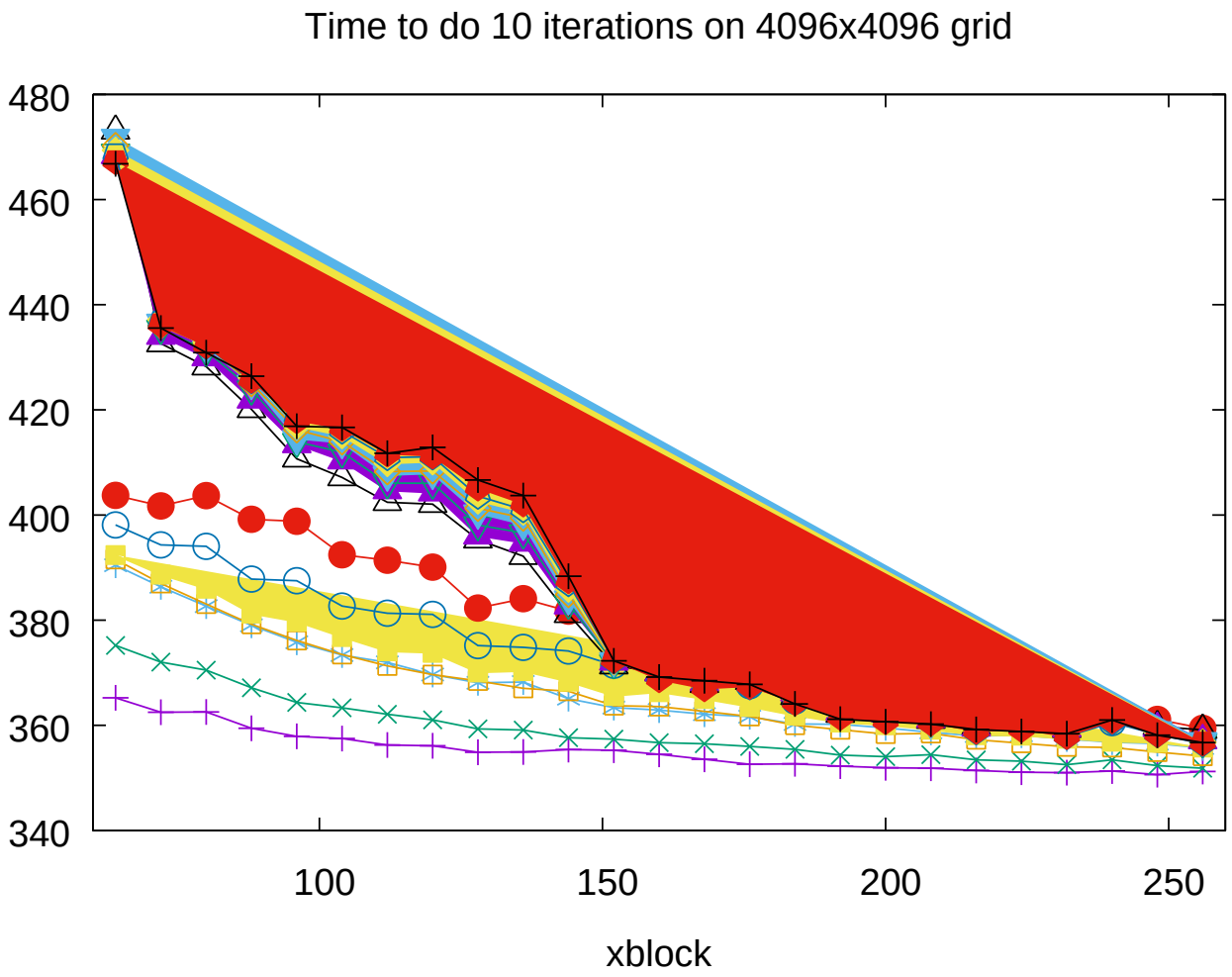
| Solution | Time in milliseconds |
|---|---|
| Xeon CPU, 1 thread | 48954 |
| Xeon CPU, 12 threads | 1767 |
| Xeon Pi coprocessor | 1409 |
| Nvidia Titan Z (single GPU) | 470 |

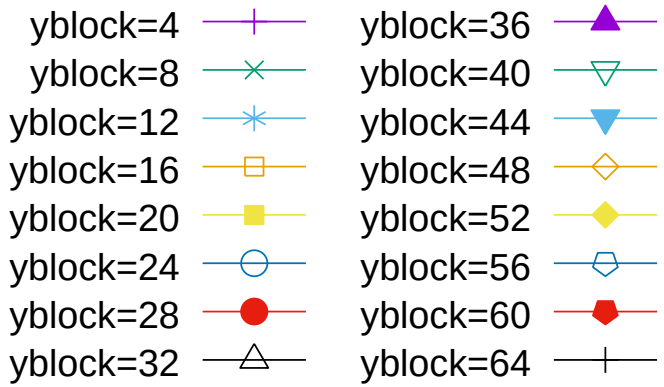
# CPU thread synchronization

- Need to wait for neighbor’s halo cells
- Simple solution: barrier
- Better solution: wait for neighbors to change flag

```
if (thread->iter > 0) {
  waitFor(thread_id-1, thread->iter);
  waitFor(thread_id+1, thread->iter);
}
```

# CPU version: Choosing tile sizes




For yblock >= 32, keep xblock >= 152

# CUDA versions

1) One thread per cell
2) Multiple cells per thread
3) Tiles in shared memory
4) Texture memory

# CUDA: One thread per cell

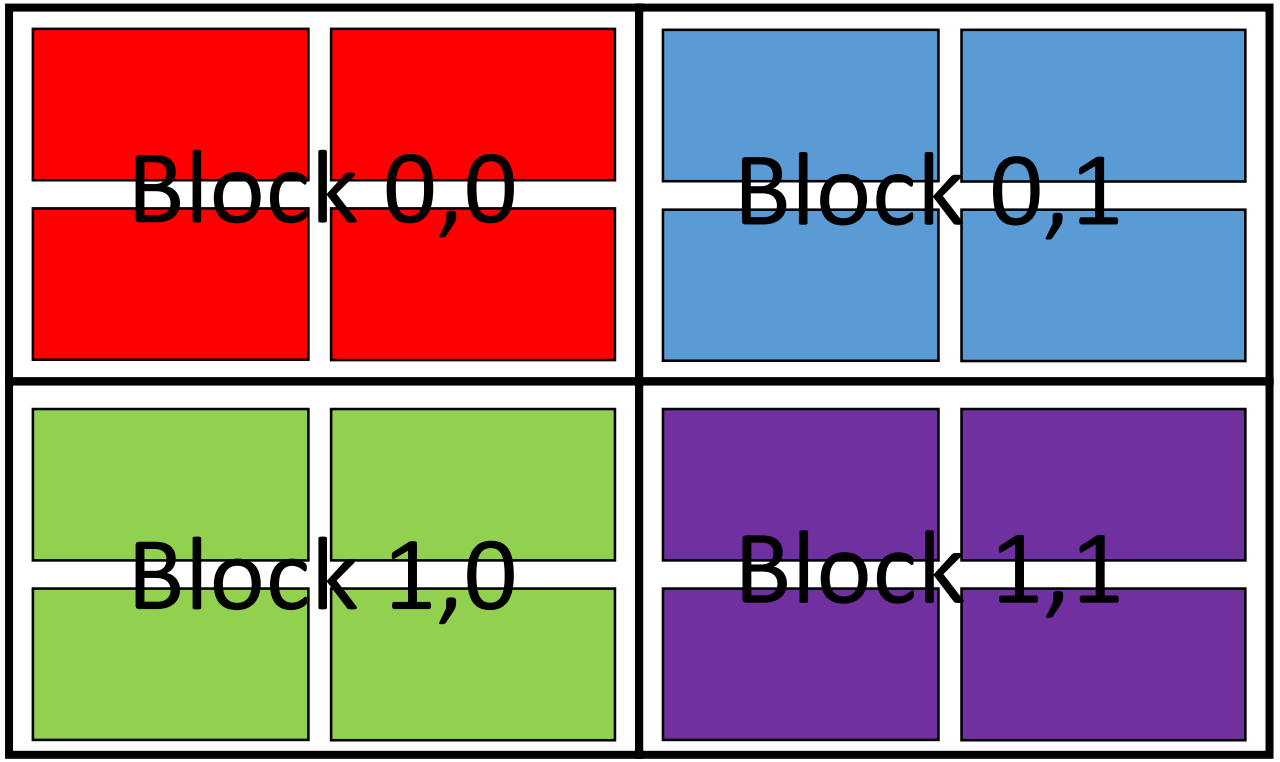
```
int y = blockIdx.y * blockDim.y + threadIdx.y;
int alternate = (1 ^ do_top_corner ^ y) & 1;
int x = 2 * (blockIdx.x * blockDim.x + threadIdx.x) + alternate;

if (x < size && y < size) {
  float tmp = 0.2f *
    (array.get(y,   x) +
     array.get(y-1, x) +
     array.get(y+1, x) +
     array.get(y,   x-1) +
     array.get(y,   x+1));

  array.set(y, x, tmp);
}
```

Best thread block size: 8 rows, 64 columns (512 threads per thread block)

Processing time: 405 ms

# CUDA: Multiple cells per thread

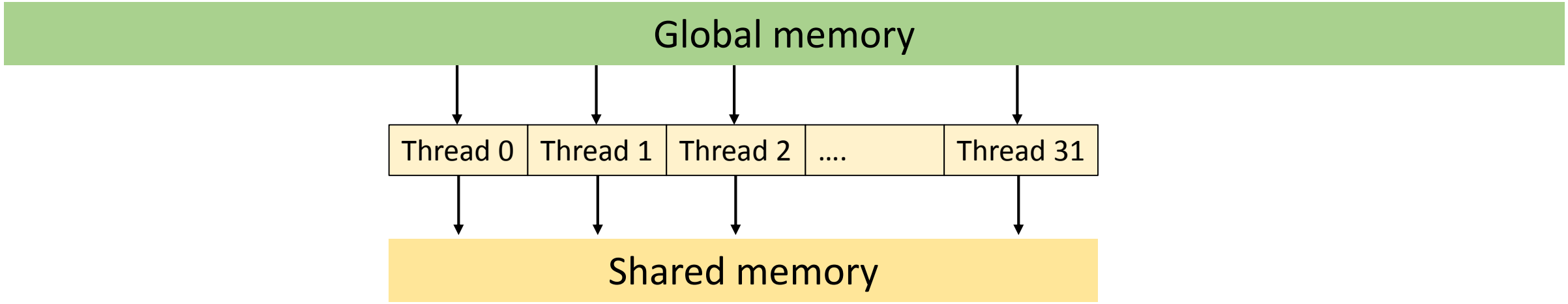
- One thread block → one tile
- One thread → multiple cells



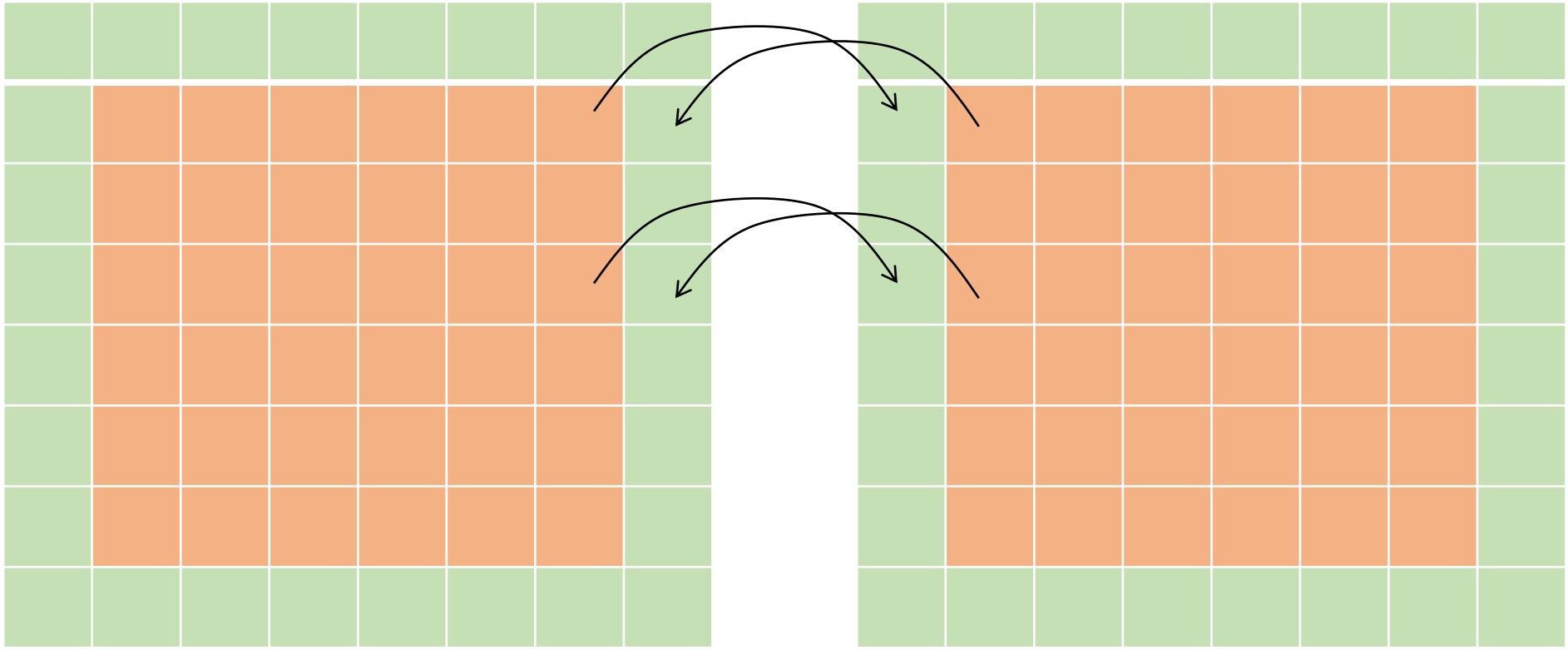

Best sizes: tiles 64x64, thread blocks 8x32
Processing time: 405 ms  (unchanged)

# CUDA: Tiles in shared memory

- Use efficient (coalesced) read from global memory into shared
- Shared memory: manually-managed cache



# CUDA: Tiles in shared memory



± 1 errors!

Each iteration: copy data+halo to tile, process it, copy just data back to global memory

Performance: 493 ms  (worse)

# CUDA: Read tile, write single cells

- Still read full tile into shared memory
- Write results directly to global memory
- No sync needed, writes overlap processing
- Performance 493 ms -> 396 ms

# CUDA: Texture memory

- Normal cache: linear
- Texture memory: 2-D
- Good performance row-major or column-major
- Textures read-only, surfaces read/write
- Performance: 691 ms (not good)

# CUDA: Page-locked memory

- Use cudaMallocHost() to allocate
- Physical memory location will not change
- Speed up xfer to & from GPU
  - 6.5 GiB/sec → 10.2 GiB/sec
  - 75 ms → 49 ms

# More work

- Speed up Xeon Phi!
  - Get compiler to vectorize
  - Find optimal tile sizes
  - Try atomic synchronization
- Implement global GPU barrier
- Multiple GPUs (2 in Titan Z, or 8 in 4 GTX 690’s)