Bandwidth Avoiding Stencil Computations

By **Kaushik Datta**, Sam Williams, Kathy Yelick, and Jim Demmel, and others

Berkeley **B**enchmarking and **Op**timization Group
UC Berkeley
March 13, 2008

http://bebop.cs.berkeley.edu
kdatta@cs.berkeley.edu

Outline

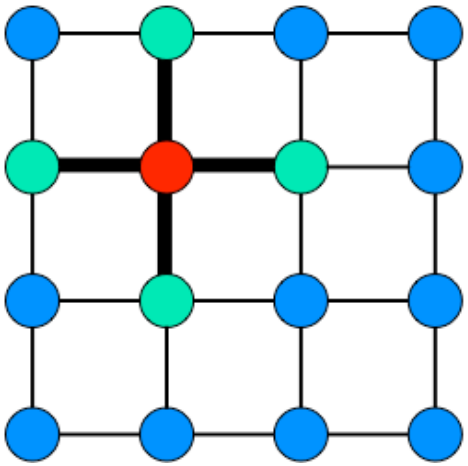
- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion

Outline

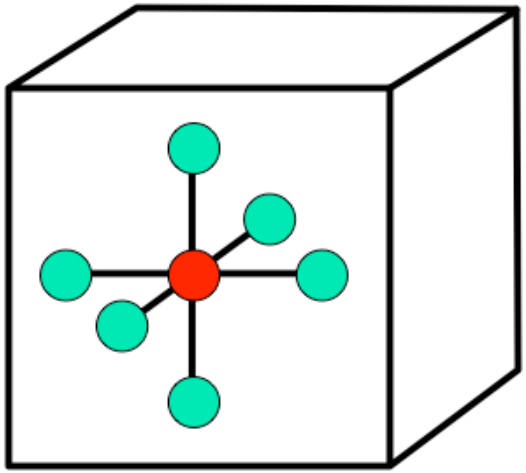
- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion

What are stencil codes?

- For a given point, a *stencil* is a pre-determined set of nearest neighbors (possibly including itself)
- A *stencil code* updates every point in a regular grid with a constant weighted subset of its neighbors ("applying a stencil")



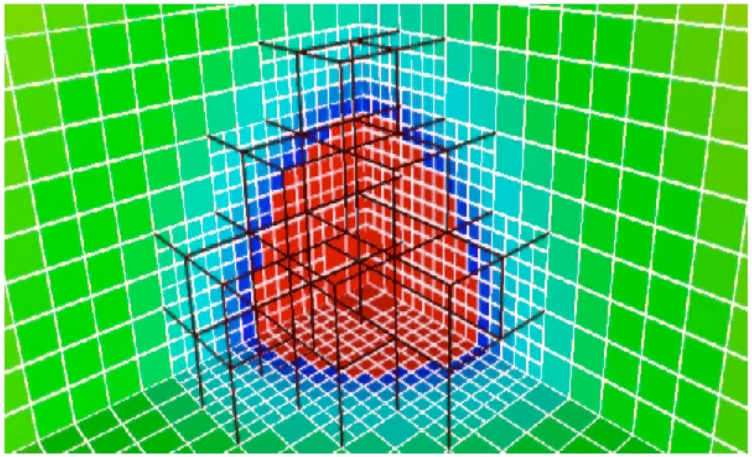
2D Stencil



3D Stencil

Stencil Applications

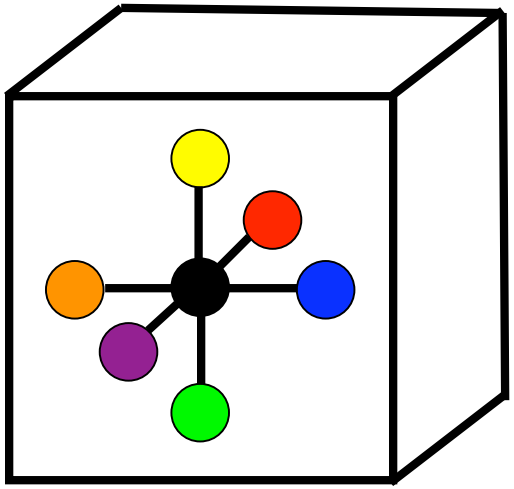
- Stencils are critical to many scientific applications:
  - Diffusion, Electromagnetics, Computational Fluid Dynamics
  - Both explicit and implicit iterative methods (e.g. Multigrid)
  - Both uniform and adaptive block-structured meshes
- Many type of stencils
  - 1D, 2D, 3D meshes
  - Number of neighbors (5-pt, 7-pt, 9-pt, 27-pt,…)
  - Gauss-Seidel (update in place) vs Jacobi iterations (2 meshes)



- This talk focuses on 3D, 7-point, Jacobi iteration

Naïve Stencil Pseudocode (One iteration)

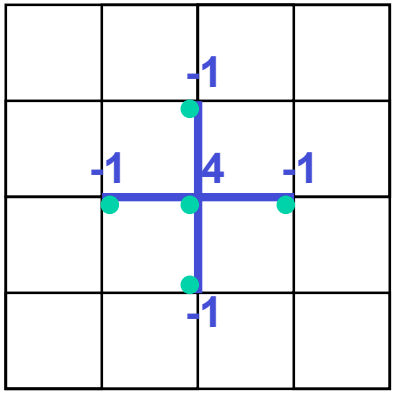
```
void stencil3d(double A[], double B[], int nx, int ny, int nz) {
   for all grid indices in x-dim {
      for all grid indices in y-dim {
         for all grid indices in z-dim {
            B[center] = S0* A[center] +
                        S1*(A[top] + A[bottom] +
                            A[left] + A[right] +
                            A[front] + A[back]);
         }
      }
   }
}
```



2D Poisson Stencil- Specific Form of SpMV

$$T = \left(\begin{array}{cccc|cccc|cccc}
4 & -1 & & & -1 & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & \\
 & -1 & 4 & & & & -1 & & & & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & -1 & & & -1 & 4 & & & & -1 & \\
\hline
 & & & & -1 & & & & 4 & -1 & & \\
 & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & -1 & & & -1 & 4 &
\end{array}\right)$$

Graph and "stencil"



- Stencil uses an *implicit* matrix
  - No indirect array accesses!
  - Stores a single value for each diagonal
- 3D stencil is analagous (but with 7 nonzero diagonals)

Reduce Memory Traffic!

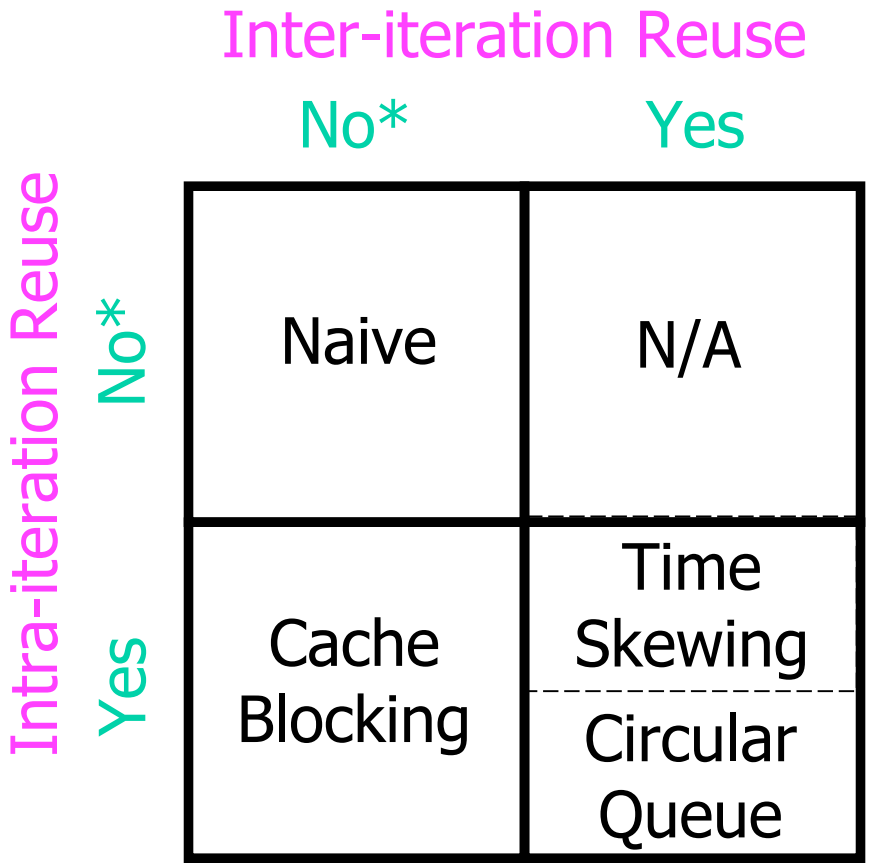
- Stencil performance usually limited by memory bandwidth
- Goal: Increase performance by minimizing memory traffic
  - Even more important for multicore!
- Concentrate on getting reuse both:
  - within an iteration
  - across iterations (Ax, A^2x, …, A^kx)
- Only interested in final result

Outline

- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion

Grid Traversal Algorithms

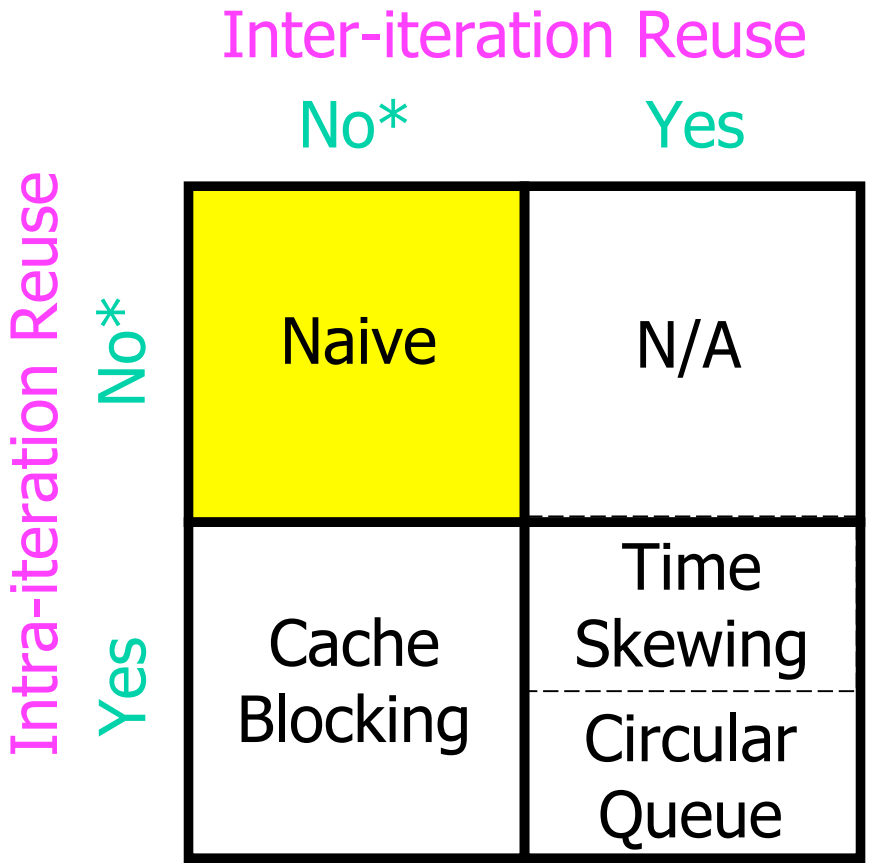
- One common technique
  - *Cache blocking* guarantees reuse within an iteration
- Two novel techniques
  - *Time Skewing* and *Circular Queue* also exploit reuse across iterations



* Under certain circumstances

Grid Traversal Algorithms

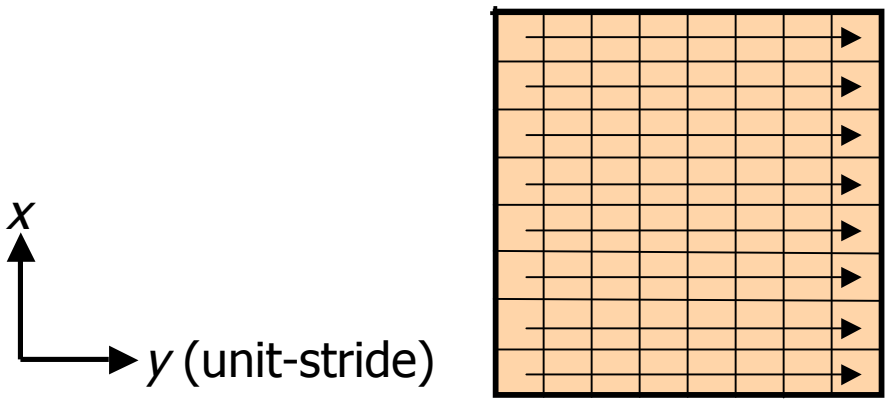
- One common technique
  - *Cache blocking* guarantees reuse within an iteration
- Two novel techniques
  - *Time Skewing* and *Circular Queue* also exploit reuse across iterations



* Under certain circumstances

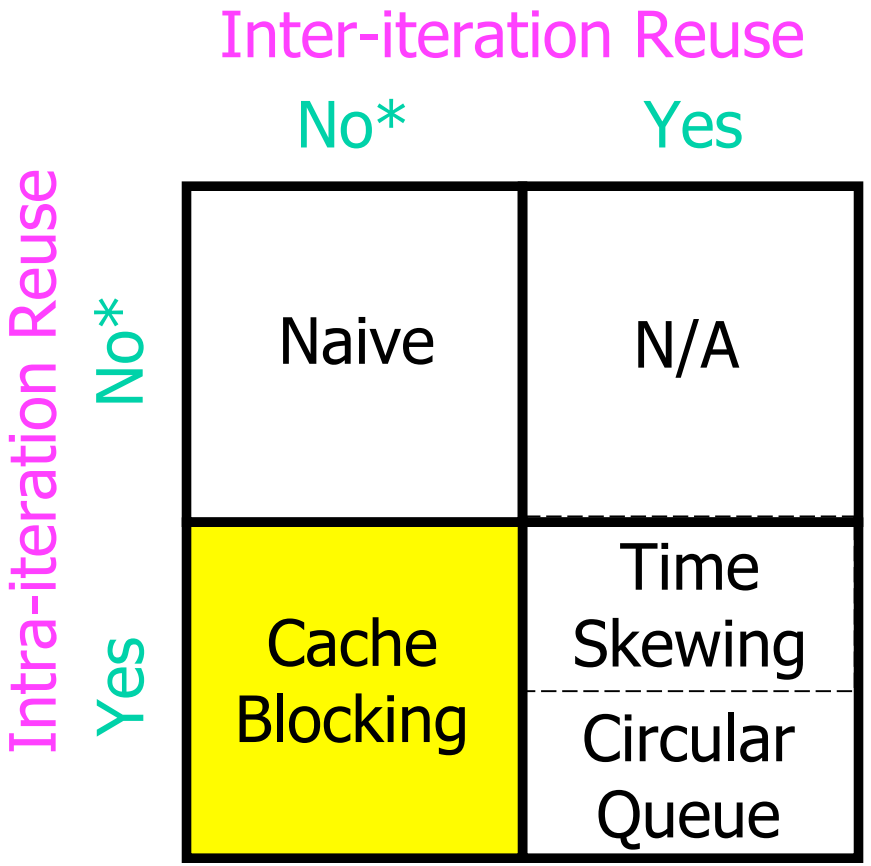
Naïve Algorithm

- Traverse the 3D grid in the usual way
  - No exploitation of locality
  - Grids that don't fit in cache will suffer



Grid Traversal Algorithms

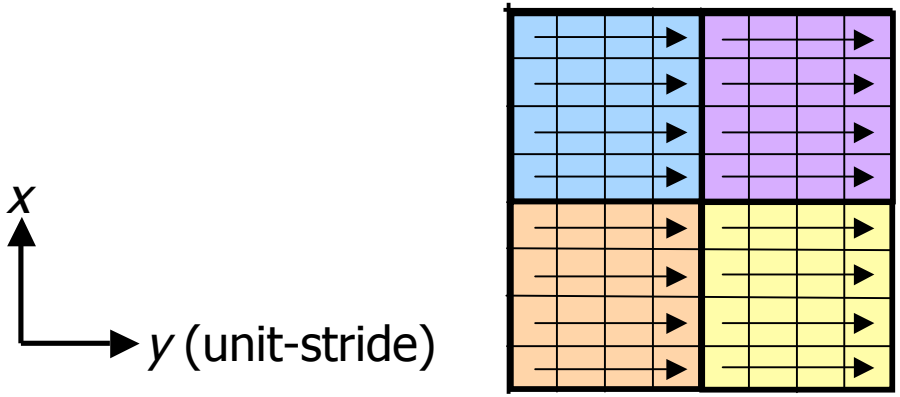
- One common technique
  - *Cache blocking* guarantees reuse within an iteration
- Two novel techniques
  - *Time Skewing* and *Circular Queue* also exploit reuse across iterations



* Under certain circumstances

Cache Blocking- Single Iteration At a Time

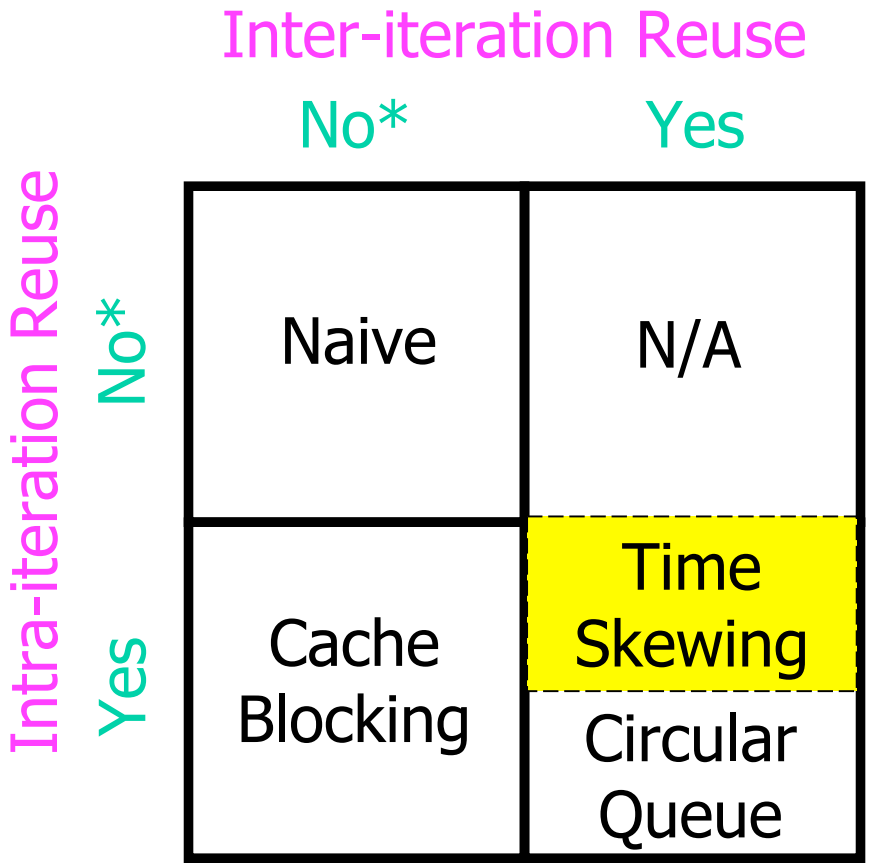
- Guarantees reuse within an iteration
  - "Shrinks" each plane so that three source planes fit into cache
  - However, no reuse across iterations



- In 3D, there is tradeoff between cache blocking and prefetching
  - Cache blocking reduces memory traffic by reusing data
  - However, short stanza lengths do not allow prefetching to hide memory latency
- Conclusion: When cache blocking, don't cut in unit-stride dimension!

Grid Traversal Algorithms

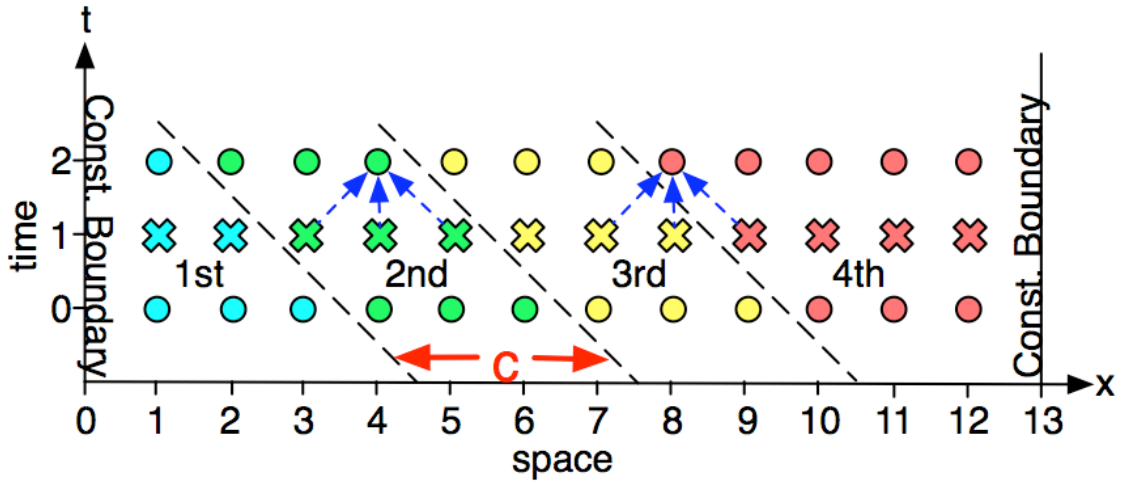
- One common technique
  - *Cache blocking* guarantees reuse within an iteration
- Two novel techniques
  - *Time Skewing* and *Circular Queue* also exploit reuse across iterations



* Under certain circumstances

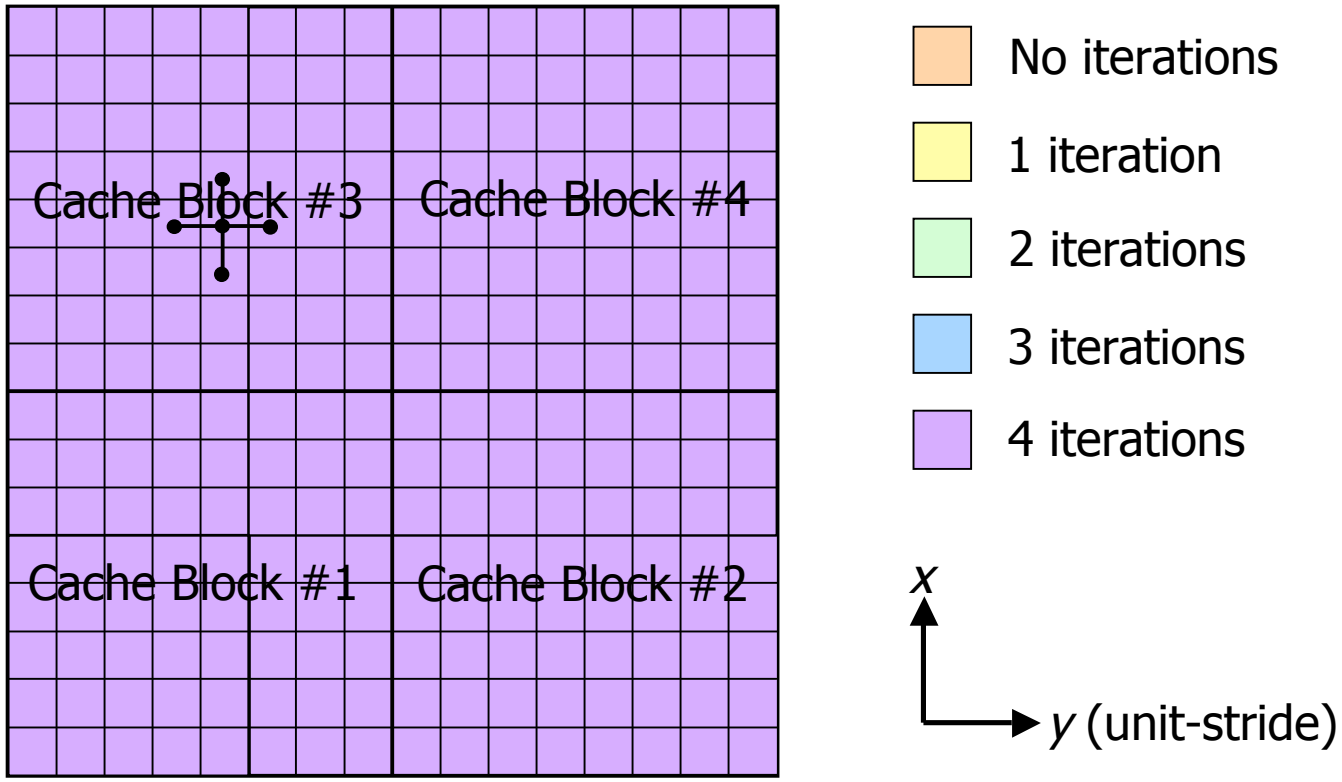
Time Skewing- Multiple Iterations At a Time

- Now we allow reuse across iterations
- Cache blocking now becomes trickier
  - Need to shift block after each iteration to respect dependencies
  - Requires cache block dimension c as a parameter (or else cache oblivious)
  - We call this "Time Skewing" [Wonnacott '00]



- Simple 3-point 1D stencil with 4 cache blocks shown above

2-D Time Skewing Animation

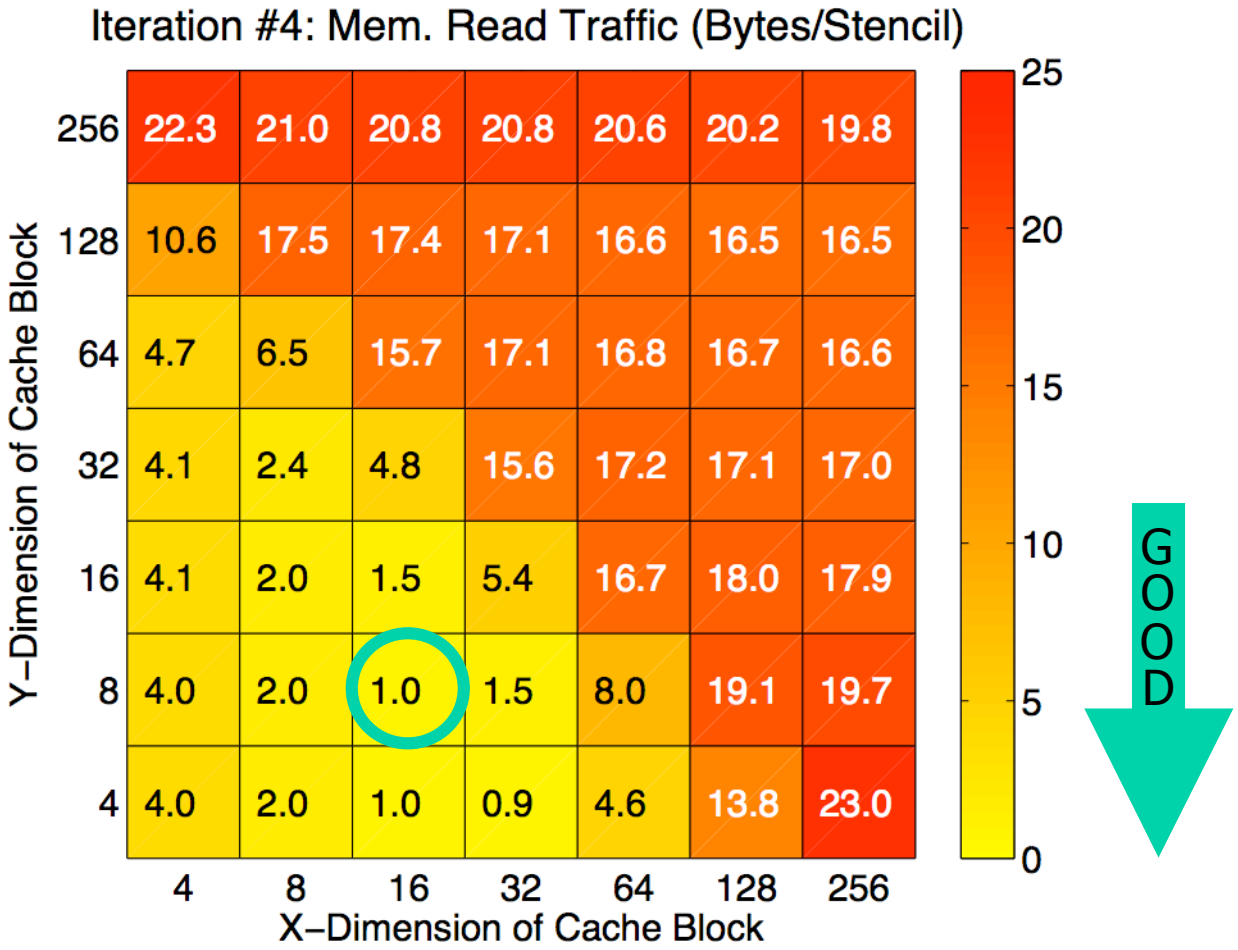


- Since these are Jacobi iterations, we alternate writes between the two arrays after each iteration

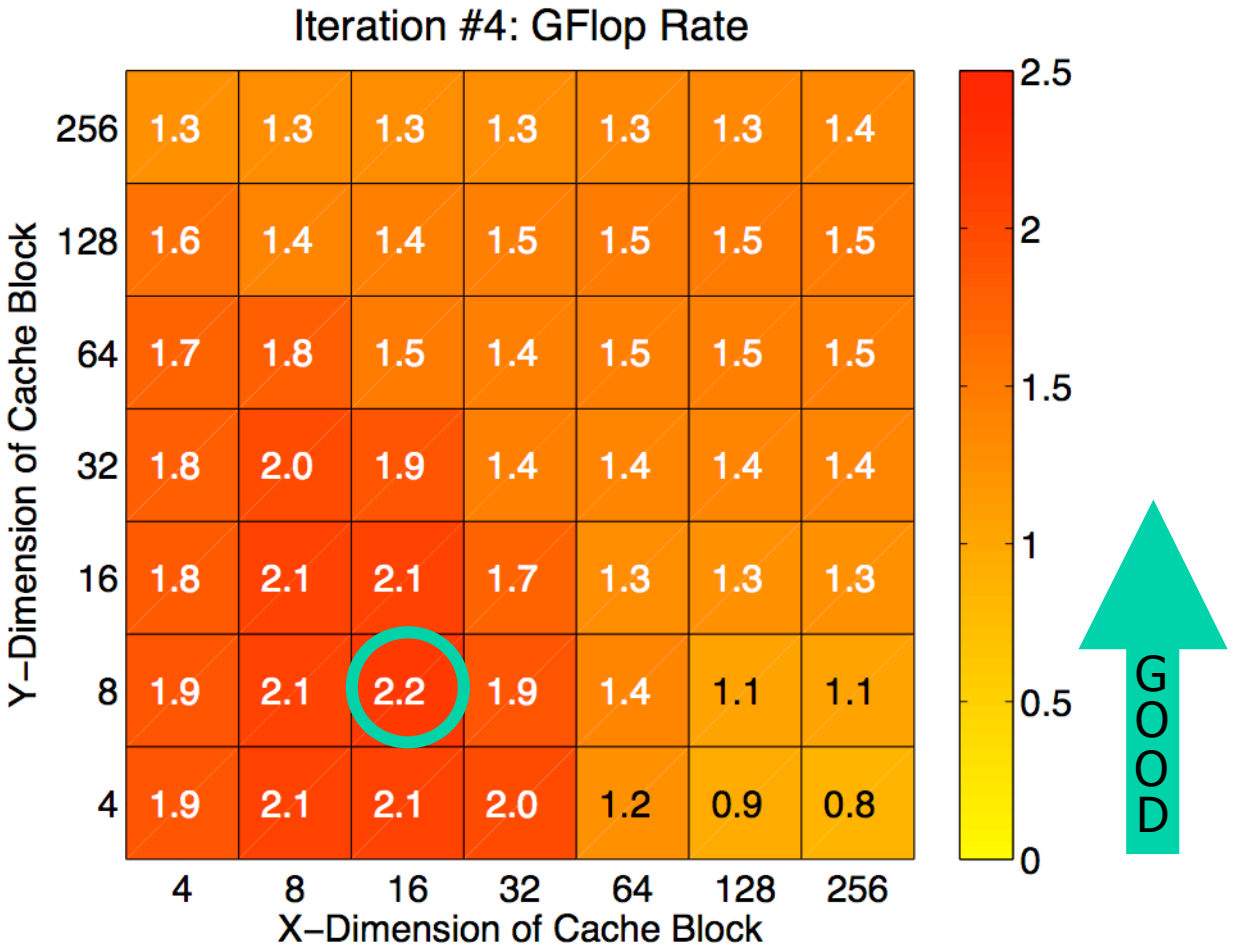
Time Skewing Analysis

- Positives
  - Exploits reuse across iterations
  - No redundant computation
  - No extra data structures
- Negatives
  - Inherently sequential
  - Need to find optimal cache block size
    - Can use exhaustive search, performance model, or heuristic
  - As number of iterations increases:
    - Cache blocks can "fall" off the grid
    - Work between cache blocks becomes more imbalanced

Time Skewing- Optimal Block Size Search



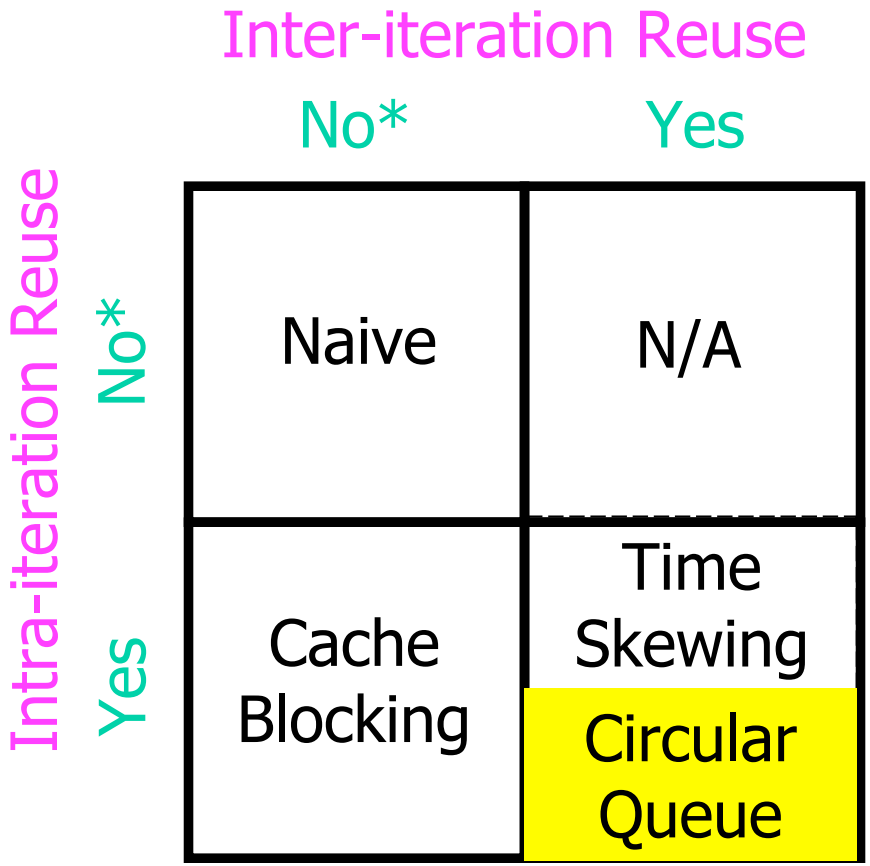
Time Skewing- Optimal Block Size Search



- Reduced memory traffic does correlate to higher GFlop rates

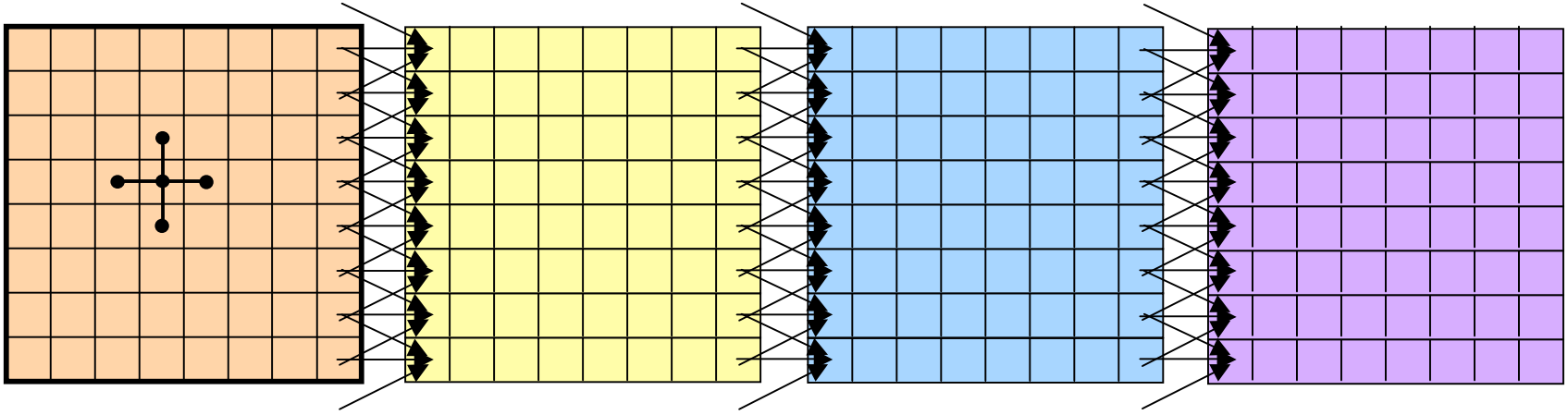
Grid Traversal Algorithms

- One common technique
  - *Cache blocking* guarantees reuse within an iteration
- Two novel techniques
  - *Time Skewing* and *Circular Queue* also exploit reuse across iterations

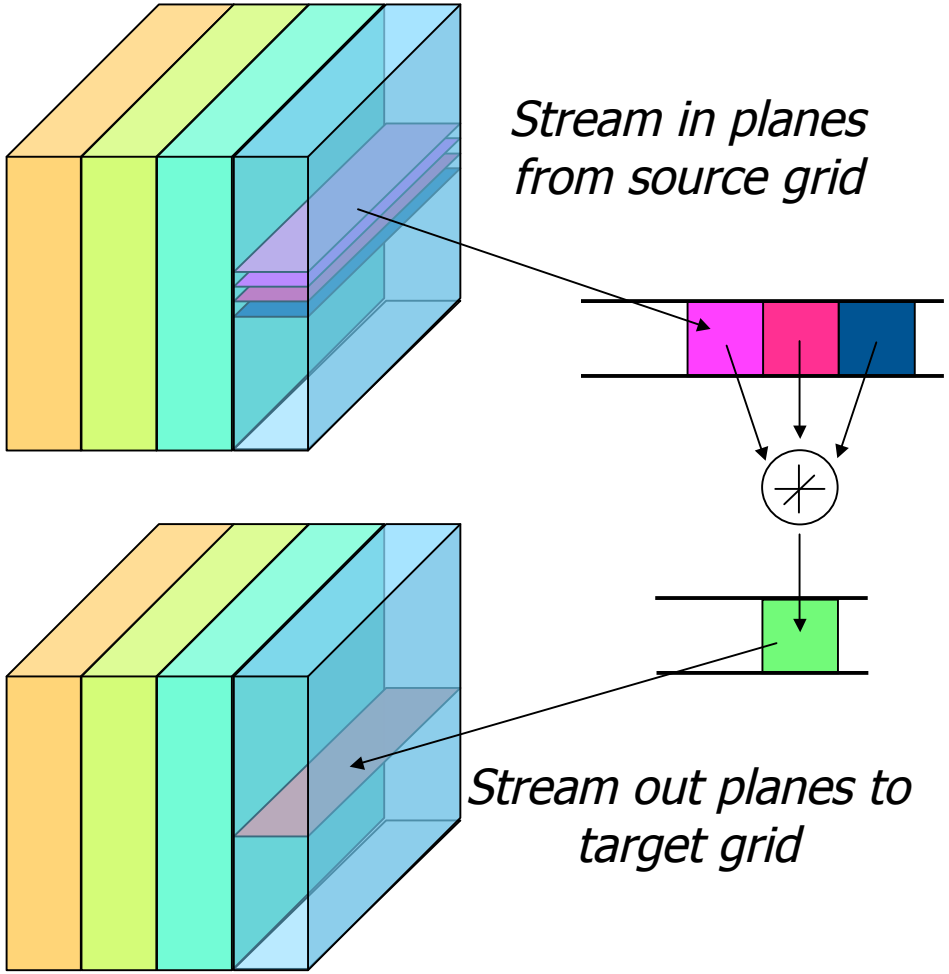


* Under certain circumstances

2-D Circular Queue Animation



Parallelizing Circular Queue



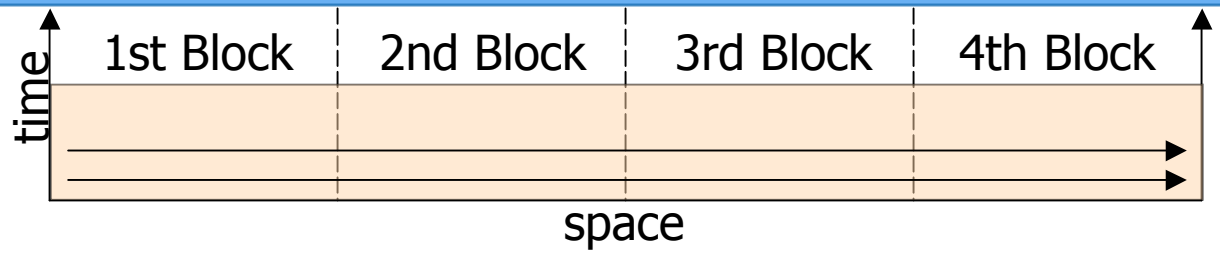
- Each processor receives a colored block
- Redundant computation when performing multiple iterations

Circular Queue Analysis

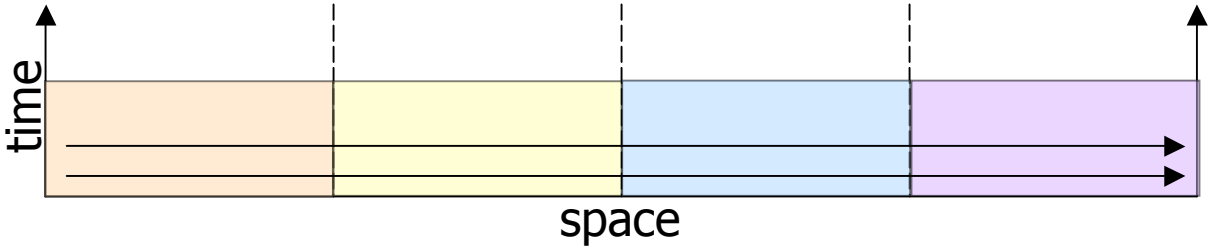
- Positives
  - Exploits reuse across iterations
  - Easily parallelizable
  - No need to alternate the source and target grids after each iteration
- Negatives
  - Redundant computation
    - Gets worse with more iterations
  - Need to find optimal cache block size
    - Can use exhaustive search, performance model, or heuristic
  - Extra data structure needed
    - However, minimal memory overhead

Algorithm Spacetime Diagrams

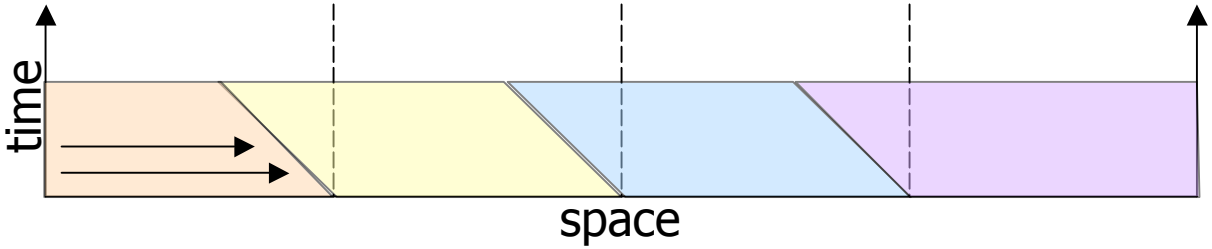
Naive



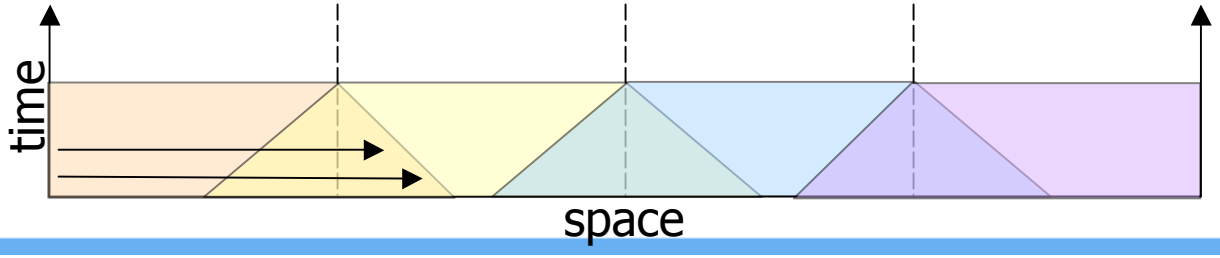
Cache Blocking



Time Skewing



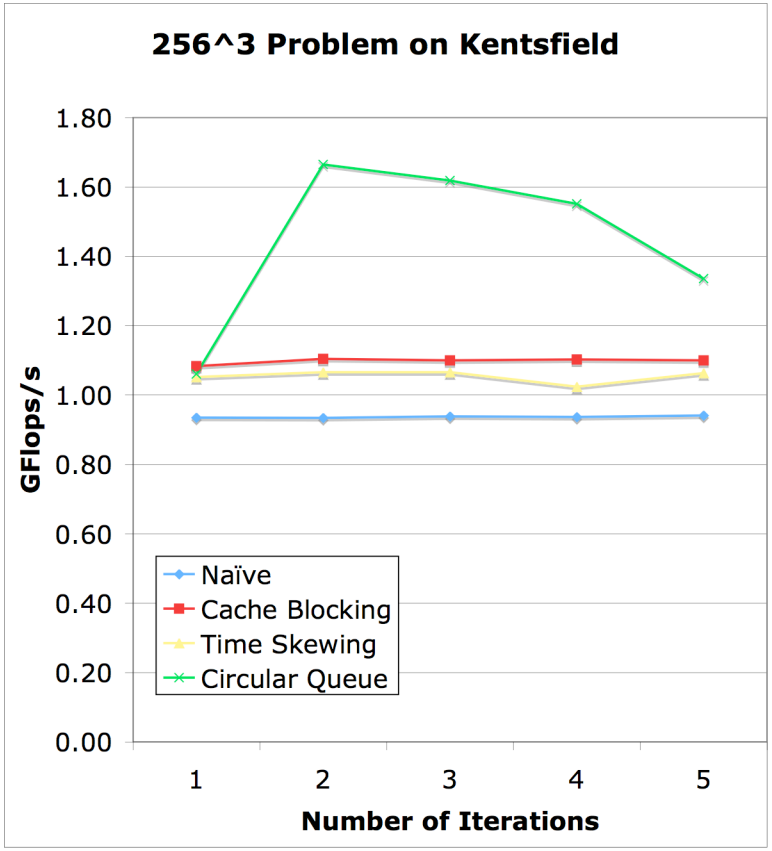
Circular Queue

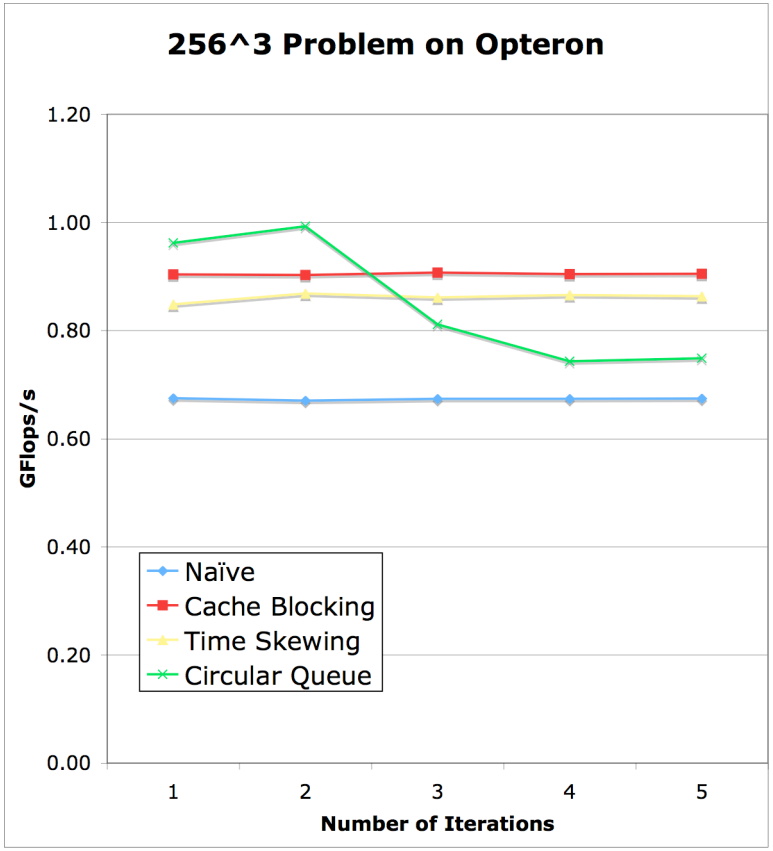


Outline

- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion

Serial Performance



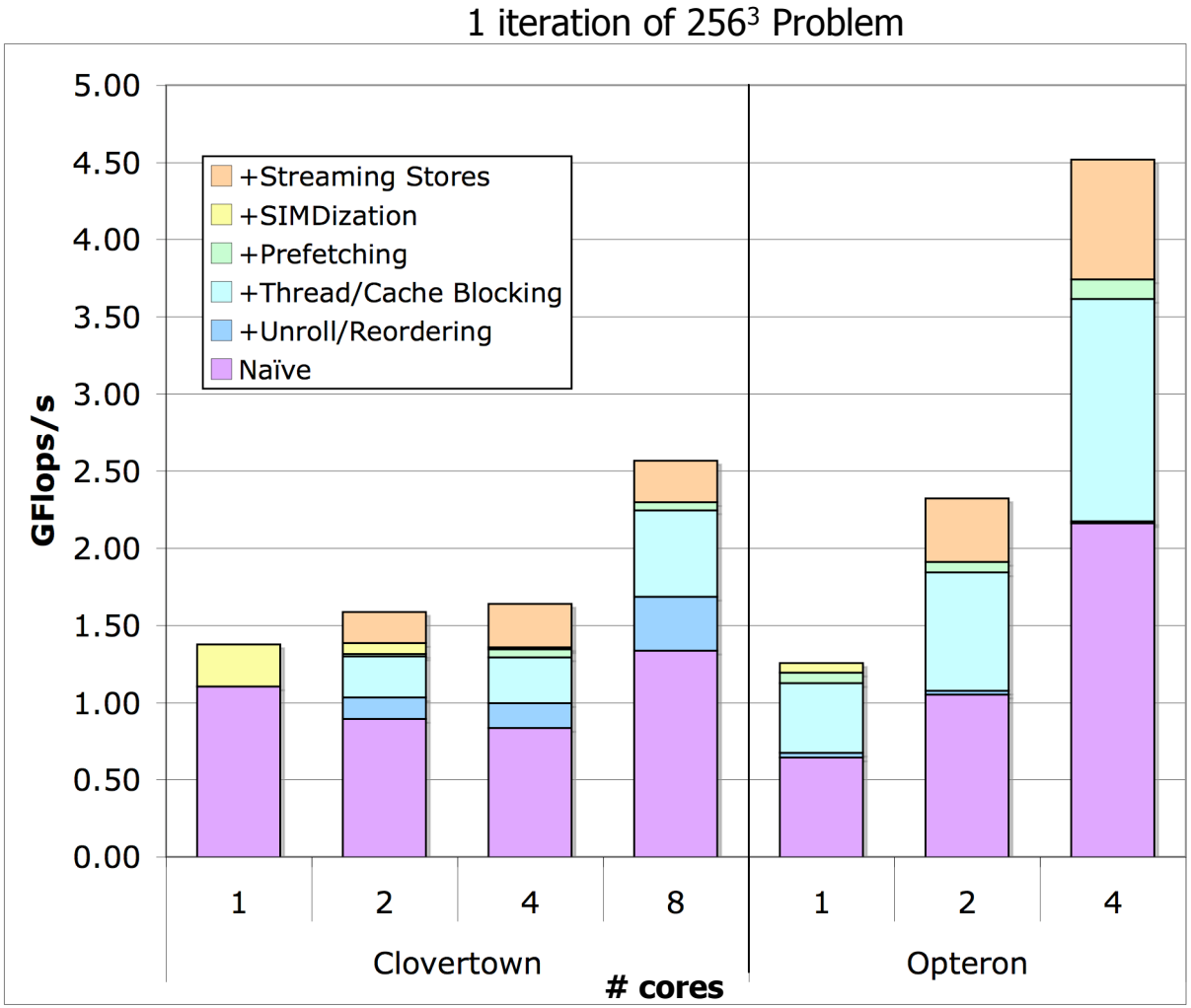


- Single core of 1 socket x 4 core Intel Xeon (Kentsfield)
- Single core of 1 socket x 2 core AMD Opteron

Outline

- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion

Multicore Performance



- Left side:
  - Intel Xeon (Clovertown)
  - 2 sockets x 4 cores
  - Machine peak DP: 85.3 GFlops/s
- Right side:
  - AMD Opteron (Rev. F)
  - 2 sockets x 2 cores
  - Machine peak DP: 17.6 GFlops/s

Outline

- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion

Stencil Code Conclusions

- Need to autotune!
  - Choosing appropriate algorithm AND block sizes for each architecture is not obvious
  - Can be used with performance model
  - My thesis work :)
- Appropriate blocking and streaming stores most important for x86 multicore
  - Streaming stores reduces mem. traffic from 24 B/pt. to 16 B/pt.
- Getting good performance out of x86 multicore chips is hard!
  - Applied 6 different optimizations, all of which helped at some point

Backup Slides

Poisson's Equation in 1D

Discretize:

$$d^2u/dx^2 = f(x)$$

on regular mesh :

$$u_i = u(i*h)$$

to get:

$$[\, u_{i+1} - 2*u_i + u_{i-1} \,] / h^2 = f(x)$$

Write as solving:

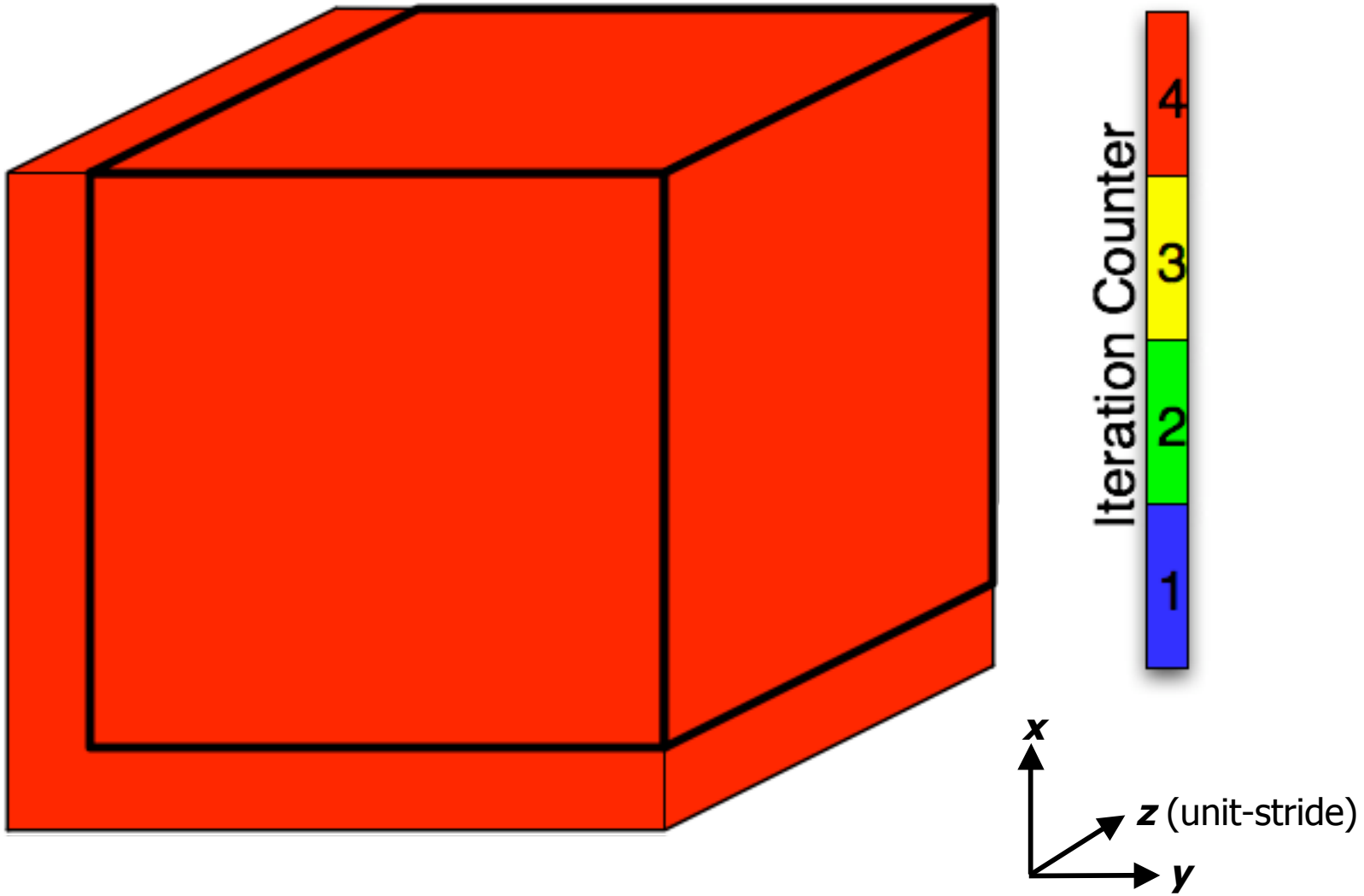
$$Tu = -h^2 * f$$

for u where

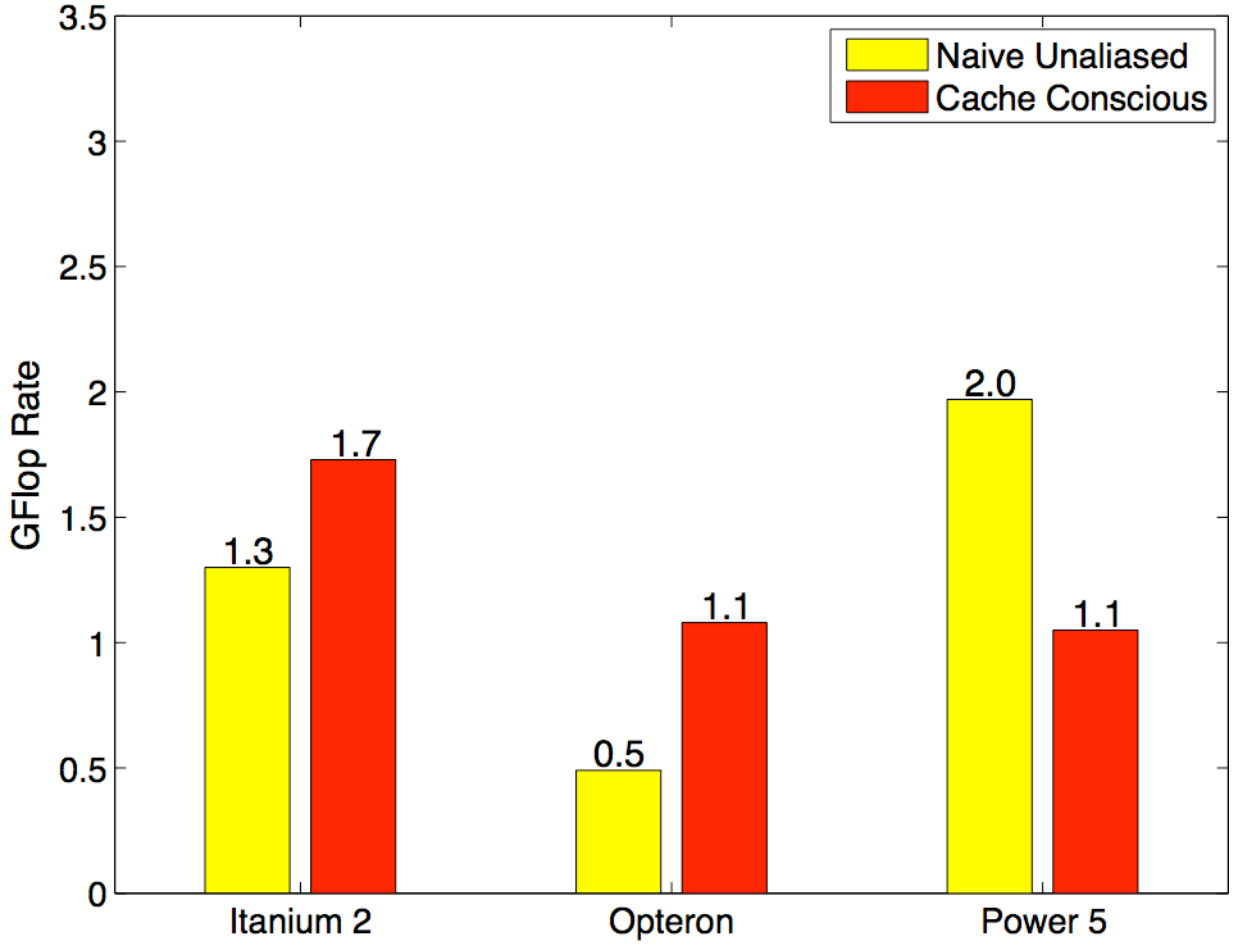
$$T = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

Graph and "stencil"

Cache Blocking with Time Skewing Animation



Cache Conscious Performance



- Cache conscious measured with optimal block size on each platform
- Itanium 2 and Opteron both improve

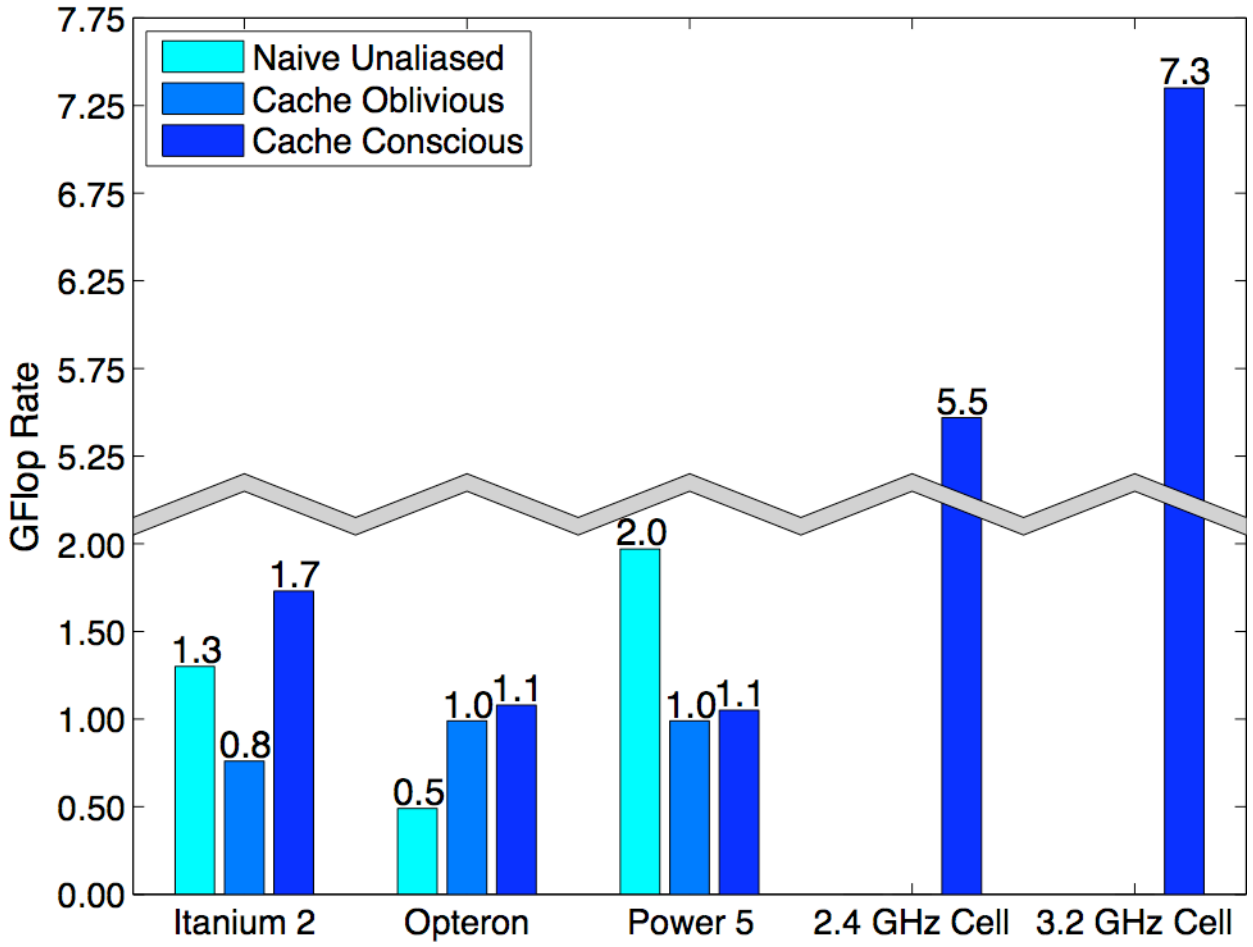
Cell Processor

- PowerPC core that controls 8 simple SIMD cores ("SPE"s)
- Memory hierarchy consists of:
  - Registers
  - Local memory
  - External DRAM
- Application *explicitly* controls memory:
  - Explicit DMA operations required to move data from DRAM to each SPE's local memory
  - Effective for predictable data access patterns
- Cell code contains more low-level intrinsics than prior code

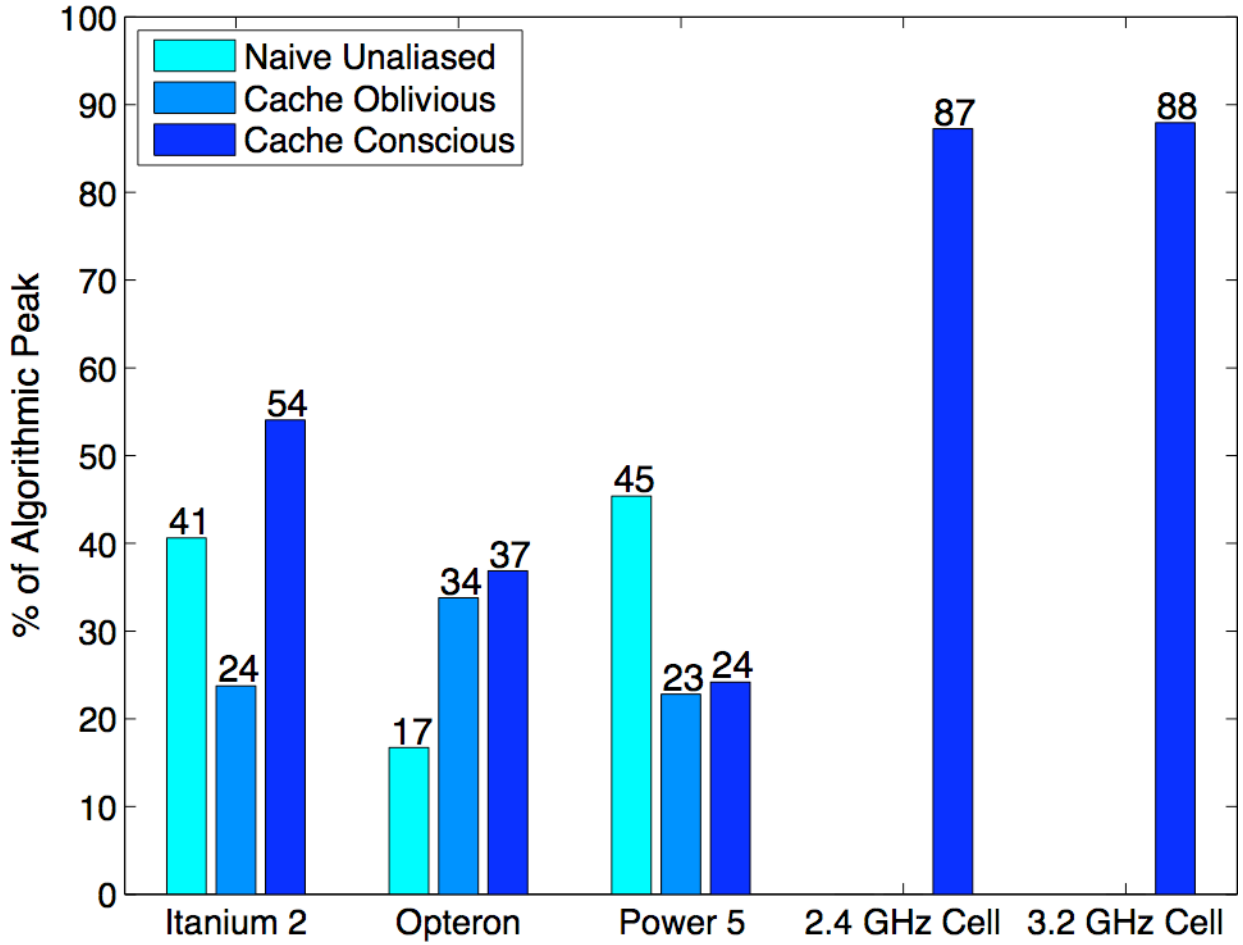
Excellent Cell Processor Performance

- Double-Precision (DP) Performance: 7.3 GFlops/s
- DP performance still relatively weak
  - Only 1 floating point instruction every 7 cycles
  - Problem becomes computation-bound when cache-blocked
- Single-Precision (SP) Performance: 65.8 GFlops/s!
  - Problem now memory-bound even when cache-blocked
- If Cell had better DP performance or ran in SP, could take further advantage of cache blocking

Summary - Computation Rate Comparison



Summary - Algorithmic Peak Comparison



Outline

- Stencil Introduction
- Grid Traversal Algorithms
- Serial Performance Results
- Parallel Performance Results
- Conclusion