CS315A/EE382B: Lecture 6

Scientific Applications

Kunle Olukotun
Stanford University

http://eeclass.stanford.edu/cs315a

Today's Outline: Scientific Apps

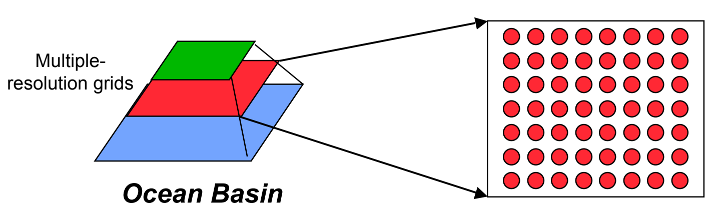
- The OCEAN multi-grid simulation benchmark
- Dense matrix kernels:
  - Matrix-vector multiply
  - Matrix-matrix multiply
  - Gaussian elimination/LU decomposition
- SPLASH-2

Sample Algorithms & Applications

- We have mostly talked about programming at a high level
  - Parallelism of basic tasks
  - Parameters of algorithm analysis
- Now let's take our analysis tools and use them!
  - Regular, dense matrix applications
  - Next lecture: applications with irregular data structures and flow

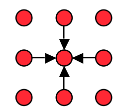
The OCEAN Benchmark

- Ocean simulates an ocean basin!
  - Basin consists of a discretized grid of points
  - Each point is a record with several parameters:
    - Temperature, salinity, current flow, etc.
  - Simulate changes over discrete steps of time
  - Multiple grids are used to adjust spatial resolution dynamically
    - Match resolution to rate-of-change of currents

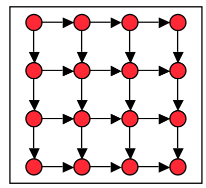


The Sequential Algorithm

- The core is a partial differential equation solver
  - Works by estimating an answer and iterating to the solution
  - Multiple grids are used to select:
    - Fast-coarse early solution approximation
    - Slow-fine final adjustments to the solution
  - Iteration cycle adjusts using NSEW neighbor information
    - Keeps repeating until solution convergence



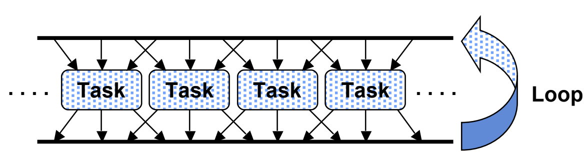
- Sequential program loops along rows
  - A top-left to bottom-right dependence
  - Not good for parallel versions!
  - Can we do better?



Algorithm Modifications

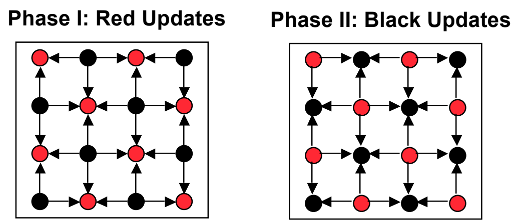
- This PDE solution method is *approximate*
  - Relies on *recent* inputs to speed convergence
  - Iteration (N:*i*, W:*i*, S:*i-1*, E:*i-1*) is just a convenient selection
  - But others are valid, too

- One obvious possibility: (N:*i-1*, W:*i-1*, S:*i-1*, E:*i-1*)
  - Eliminates all dependencies within an iteration
  - Makes each iteration *completely parallel*



Red-Black Gauss-Seidel

- Further refinement leads to “red-black” approximation
  - Basic parallel version requires odd-i and even-i grids
    - Double the memory requirements
  - Split each iteration into two phases to avoid this
    - “Red” & “black” checkerboard squares alternate
  - Now just need two barriers/timestep



Algorithm Analysis

- We can now analyze the parallel algorithm:
  - Computation/communication ratio
  - Type(s) of communication required
  - All in terms of n (side of grid) and P (number of processors)
    - Note that in past, we used N = *size* of grid (n^2 here)
  - Concurrency: Maximum number of useful processors
    - Only in terms of n, since this sets P

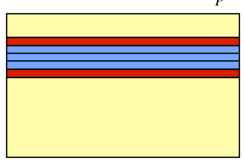
- Concurrency for red-black Ocean is simple!

Communication Analysis

- Ocean just uses nearest-neighbor communication
  - Cheap and easy to implement on any shared parallel processor
- We can choose a 1-D or 2-D partitioning
  - Has no impact on computation
  - But affects communication:

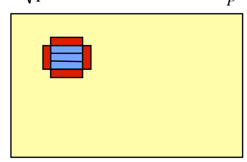
Comms/Stripe: $2n$

Elements/Stripe: $\frac{n^2}{p}$



Comms/Block: $4\frac{n}{\sqrt{p}}$

Elements/Block: $\frac{n^2}{p}$

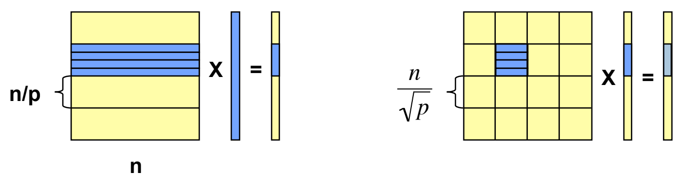


Computation-Communication Ratio

- We can now calculate computation-communication ratios:
  - 1-D, "striped" division: $\frac{n}{p}$
  - 2-D, "blocked" division: $\frac{n}{\sqrt{p}}$ ← **Clearly better!**
- The 2-D division version is better
  - Offers more concurrency: n^2 instead of n
  - Less communication for same computation
  - Also lends itself well to 4-D array blocking techniques
    - Has been shown to speed up better than 50%
    - 4-D arrays avoid *false sharing* at edges of blocks

Matrix-Vector Multiply

- One of the simplest linear algebra functions
  - Requires simple data manipulation
  - Requires some reduction
- Two ways to allocate by data:
  - 1-D division, by row on input (or chunk of output)
  - 2-D division, blocked on input



Computation & Communication Analysis

- Just one MAC (multiply-accumulate) per matrix element
- Only need to communicate along rows
  - Nothing for 1-D
  - Reduction sum communication for 2-D

Comms/Stripe:	MACs/Stripe:	Comms/Block:	MACs/Block:
None	$\frac{n^2}{p}$	Reduction	$\frac{n^2}{p}$

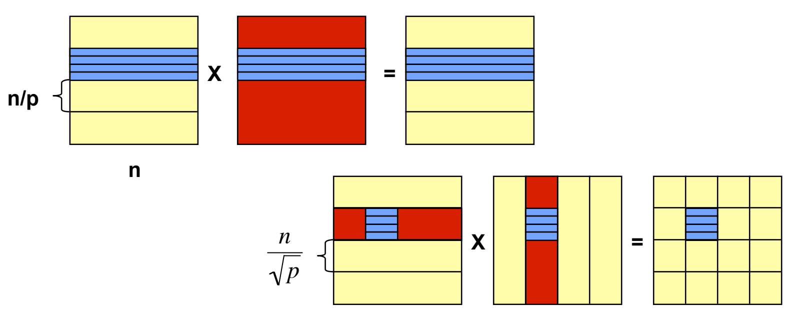
- So why even bother with a blocked implementation?

Concurrency Analysis

- 2-D offers more possible *concurrency*
  - 1-D only lets you use n processors
  - 2-D lets you use up to n^2 processors!
    - At the expense of reduction trees for each output
  - This is a common advantage of 2-D division!
- 2-D may also be a better choice if you're multiplying the matrix before or after this step

Matrix-Matrix Multiply

- Now let's add another dimension to an input
- Two basic ways to allocate by data/execution:
  - 1-D division, by row on input/output
  - 2-D division, blocked on output



Computation & Communication Analysis

- n MACs per matrix element
- Blocking now improves both:
  - *Concurrency:* n to n^2
  - *Communication:* Each block only needs 1/sqrt(p) of each input matrix

Comms/Stripe:	**MACs/Stripe:**	**Comms/Block:**	**MACs/Block:**
n^2	$\frac{n^3}{p}$	$2\frac{n^2}{\sqrt{p}}$	$\frac{n^3}{p}$

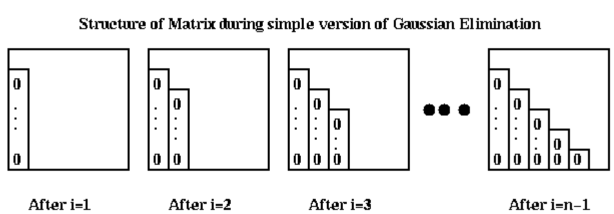
Computation-Comm. Ratio: $\frac{n}{p}$

Computation- Comm. Ratio: **Better!** → $\frac{n}{\sqrt{p}}$

Review of Gaussian Elimination for solving Ax=b

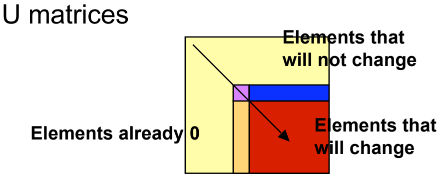
- Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows



LU Decomposition

- A critical linear algebra function is solving systems of equations
  - Gaussian elimination is the basic technique
  - Decomposition of A into L & U matrices

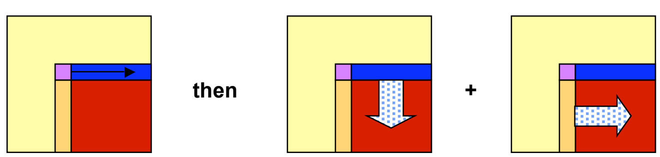


- Basic algorithm:
  - Loop n times through algorithm (iterator k)
  - Divide A[k,k] into all remaining values in its row (k+1 to n)
    - Note that this uses many potentially expensive division ops
  - Subtract row k • A[row, k] from all remaining rows (k+1 to n)
    - This is just a lot of MAC operations

 17

Communication Patterns

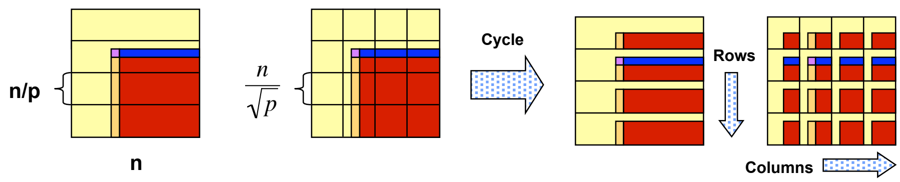
- Normally divided up by mapping processors to areas of A
- Communication is mostly symmetric:
  - k^{th} column must be broadcast across to right
  - k^{th} row must be broadcast down to bottom
    - Subtract out the product of these two communications
  - Plus A[k, k] must be broadcast down row k first



 18

Work/Data Division Patterns

- Can divide up in 1-D striped or 2-D blocked arrangements
  - 1-D works OK both with columns OR rows
    - Must communicate in both directions, anyway
  - 2-D offers n^2 concurrency, while 1-D only offers n
- Block-cyclic distribution is necessary for **load balancing**
  - Only the lower-right corner of the array is active
  - Need to spread this corner out among processors
  - B-C reduces load imbalance to no more than 1 row/column



Why SPLASH-2?

- SPLASH
  - Small number of programs
  - Programs not scalable
- SPLASH-2
  - Broader range of coverage, improved algorithms
  - Designed for scalability
- Goals of paper
  - Characterization of SPLASH-2 programs
  - Methodology for architectural studies

The SPLASH-2 Applications

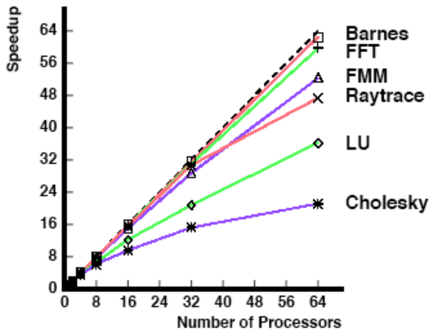
	Code	New	Domain	Representative of	Multiple Data Sets
Applications	Barnes		Astrophysics	Hierarchical N-body methods	✓
	FMM	✓	Astrophysics	Hierarchical N-body methods	✓
	Water-Nsq		Chemistry	N-body methods	✓
	Water-Sp	✓	Chemistry	N-body methods	✓
	Radiosity	✓	Graphics	Hierarchical radiosity	
	Raytrace	✓	Graphics	Optimized ray tracing	
	Volrend	✓	Graphics	Volume rendering	✓
	Ocean		CFD	Regular grid iteration	✓
Kernels	LU	✓	Radar cross section	Dense matrix factorization	✓
	Cholesky		Finite element	Sparse matrix factorization	✓
	FFT	✓	Signal processing	Convolution/Transform	✓
	Radix	✓	Sorting	Sorting	✓

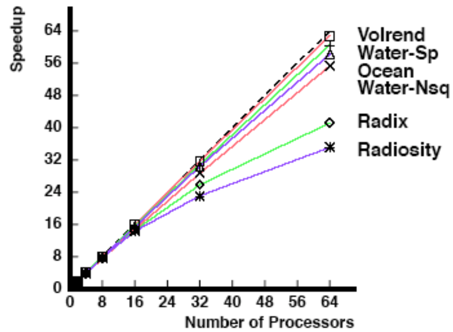
SPLASH-2 Characterization

- Axes of Characterization
  - Concurrency
  - Temporal Locality and Working Sets
  - Communication-to-Computation Ratio and Traffic
  - Spatial Locality
- Effects of
  - Machine model
  - Data set size
  - Inherent versus practical considerations

Concurrency

- PRAM model
- Speedups



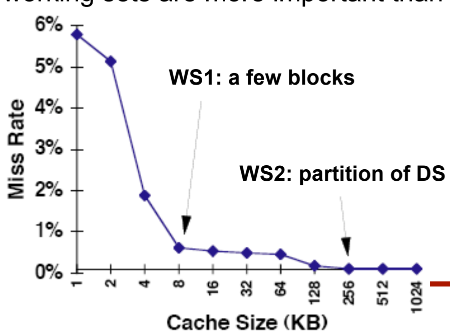


- Methodological consideration?

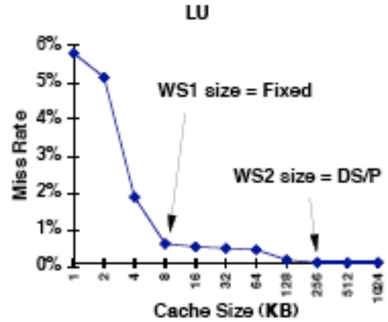
Temporal Locality and Working Sets

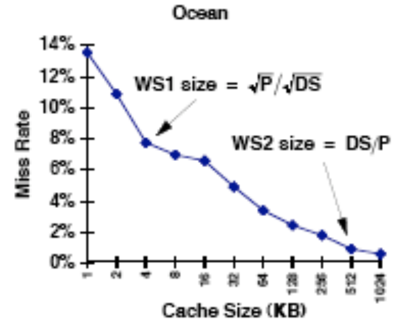
- Motivation
  - Temporal locality ⇒ miss rate ⇒ performance
- Working sets
  - Working sets denoted by knees in miss rate curve
  - Hierarchy of working sets
  - Some working sets are more important than others

LU



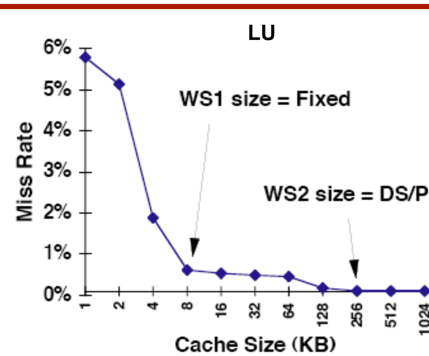
Example Working Sets





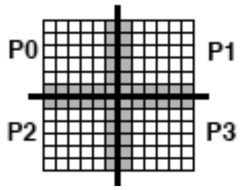
- Characteristics of working sets
  - Parameter dependent
  - Grow at different rates
  - May not be well-defined
  - How should you measure miss rates?

Working Set Methodological Implications



Communication-to-Computation Ratio and Traffic

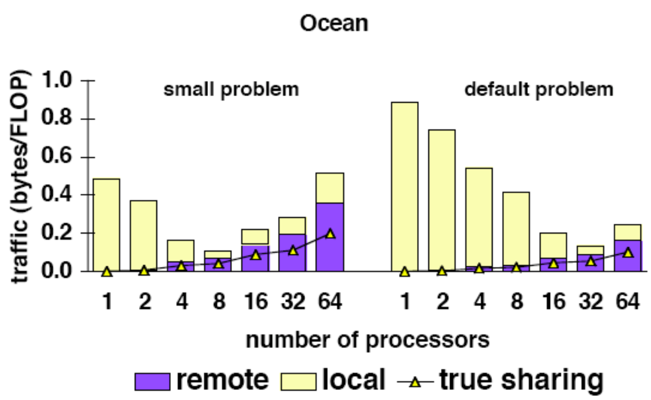
- Comm-to-comp ratio: inherent traffic ⇒ lower bound



Communication ∝ perimeter
Computation ∝ area

- Why do we care?
- Components of traffic
  - Inherent traffic
  - Capacity traffic
  - Artifactual traffic

Traffic Example



- Characteristics of traffic
  - Parameter dependent (procs, problem size, ...)
  - Composition changes with parameters

Traffic: Implications

- Methodological implications

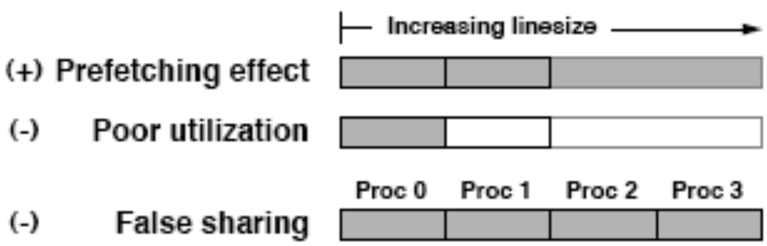
- Why do Radix and FFT have traffic graphs that flatten out with processor count?

 31

Spatial Locality

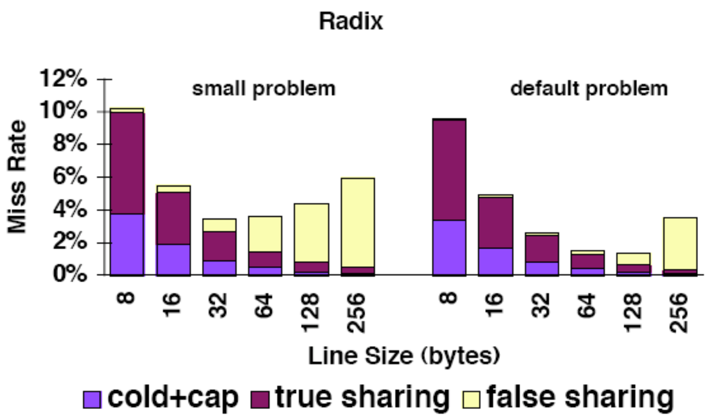
- Motivation
  - Spatial locality ⇒ miss rate ⇒ performance
  - Choice of line size

- Linesize effects



 33

Spatial Locality Example



- Characteristics of spatial locality
  - Parameter dependent (procs, problem size, ...)
  - Miss composition changes with parameters

Spatial Locality Implications

- Methodological implications

Summary and Look Ahead

- Dense matrix kernels offer interesting tradeoffs between:
  - Communication
  - Computation
  - Maximum concurrency
- SPLASH-2
  - Need to understand how parameters affect results
  - Conclusion = f(appl,prob size,cache size,line size,# procs,...)
    - Some parameters are easy to prune
  - Important to understand program behavior!
- More applications
  - Scientific applications with irregular data structures and flow
  - Commercial applications
    - Read paper