

Lecture 9: The Fast Fourier Transform

November 27, 2020

*Thanks to Jim Demmel for many slides

Remainder of the Semester

- Today:
 - Lecture: FFT
 - Lab: Hybrid Parallel Programming
- December 4
 - Lecture: Particle Methods
 - Lab: Particle Methods
 - Homework 2 due, Homework 3 posted
- December 11
 - Lecture: Graph partitioning
 - Lab: Graph partitioning (MATLAB)
- December 18
 - Lecture: High performance machine learning
 - Lab: Mapreduce/Spark tutorial (python)
- January 8
 - Lecture: GPU programming
 - Lab: Semester review
 - Homework 3 due
- Exam Period: January 11 – February 14

Homework 1

- Grades have been posted
- Most writeups were very, very nicely done
- Some data races in code – be sure to check that residual after fixed number of iterations is the same regardless of how many threads you use!
- Reminders for future homeworks
 - In the writeup, it is always a good idea to **try more than one problem size** for your experiments
 - Make sure to be **clear about what machine you ran on**, what SLURM parameters you used so that the experiments are reproducible
 - A key point of the writeup is to **think about and explain the scaling behavior you see!** (Resource contention? Communication cost? Problem size too small? Etc.)
- README file should include instructions on how to compile and run your code, any parameters that need to be input, what the code is actually doing (solving a problem with what as the right-hand-side, etc.)

Outline for today

- Definitions
- A few applications of FFTs
- Sequential algorithm
- Parallel 1D FFT
- Parallel 3D FFT
- Autotuning FFTs: FFTW

Definition of Discrete Fourier Transform (DFT)

- Let $i = \sqrt{-1}$ and index matrices and vectors from 0
- The (1D) **DFT** of an m -element vector v is:

$$F * v$$

where F is an m -by- m matrix defined as:

$$F[j, k] = \bar{\omega}^{(j*k)}, \quad 0 \leq j, k \leq m - 1$$

and where $\bar{\omega}$ is:

$$\bar{\omega} = e^{2\pi/m} = \cos(2\pi/m) + i * \sin(2\pi/m)$$

- $\bar{\omega}$ is a complex number with whose m^{th} power $\bar{\omega}^m = 1$ and is therefore called an m^{th} root of unity
- E.g., for $m = 4$: $\bar{\omega} = i$, $\bar{\omega}^2 = -1$, $\bar{\omega}^3 = -i$, $\bar{\omega}^4 = 1$

Definition of Discrete Fourier Transform (DFT)

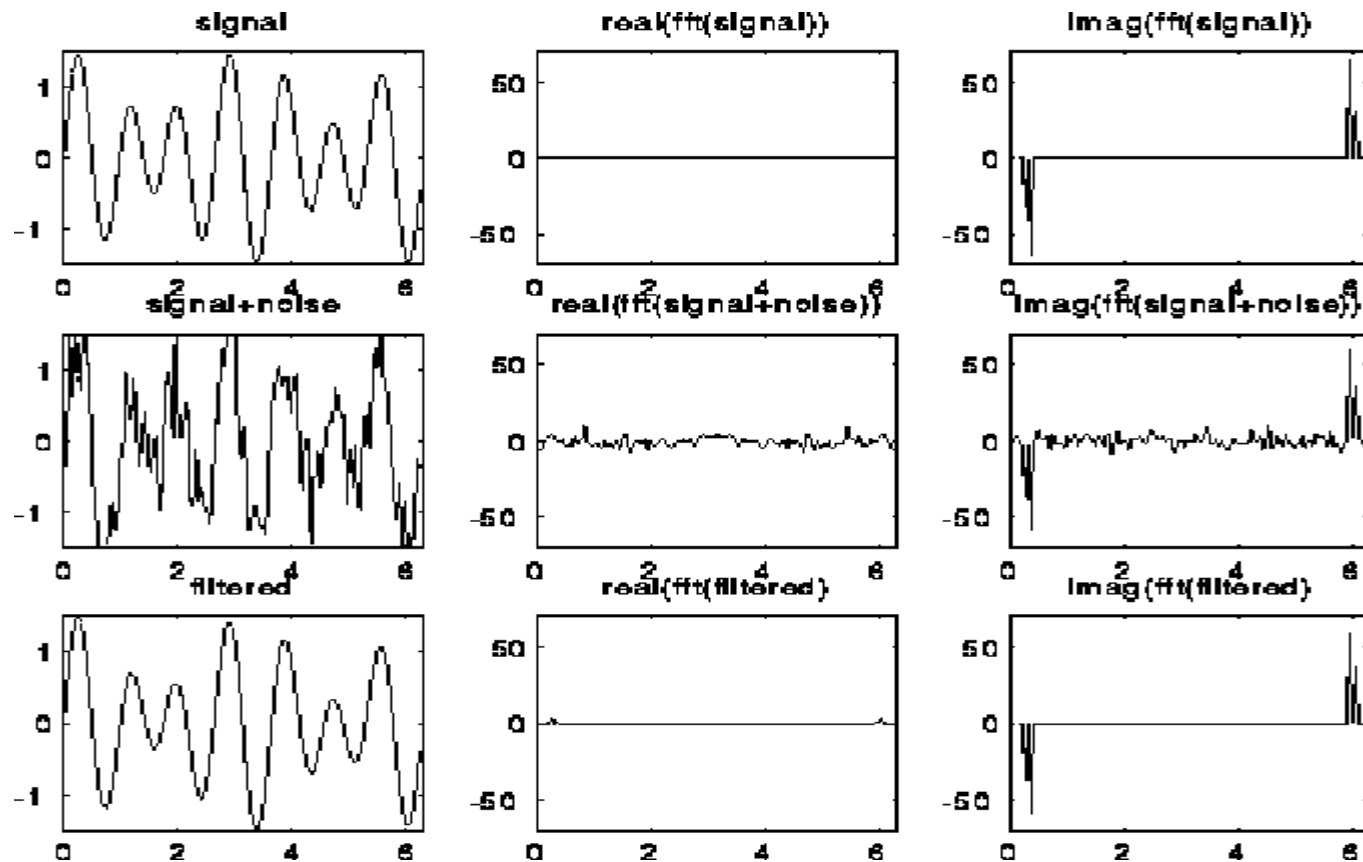
- The 2D DFT of an m -by- m matrix V is $F*V*F$
 - Do 1D DFT on all the columns independently, then all the rows
- Higher dimensional DFTs are analogous

Motivation for *Fast* Fourier Transform (FFT)

- Signal processing
- Image processing
- Solving Poisson's Equation nearly optimally
 - $O(N \log N)$ arithmetic operations, $N = \#$ unknowns
 - Competitive with multigrid
- Fast multiplication of large integers
- ...

Using the 1D FFT for filtering

- Signal = $\sin(7t) + .5 \sin(5t)$ at 128 points
- Noise = random number bounded by .75
- Filter by zeroing out FFT components $< .25$



Using the 2D FFT for image compression

- Image = $p_r \times p_c$ matrix of values
- Compress by keeping e.g., largest 2.5% of FFT components
- Similar idea used by jpeg

Recall: Poisson's equation arises in many models

$$\text{3D: } \partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 + \partial^2 u / \partial z^2 = f(x, y, z)$$

$$\text{2D: } \partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 = f(x, y)$$

$$\text{1D: } d^2 u / dx^2 = f(x)$$

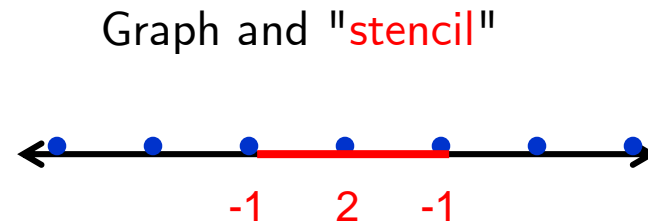
f represents the sources; also need boundary conditions

- Electrostatic or Gravitational Potential: **Potential(position)**
- Heat flow: **Temperature(position, time)**
- Diffusion: **Concentration(position, time)**
- Fluid flow: **Velocity, Pressure, Density(position, time)**
- Elasticity: **Stress, Strain(position, time)**
- Variations of Poisson have variable coefficients

Solving Poisson Equation with FFT (1/2)

- 1D Poisson equation: solve $L_1 x = b$ where

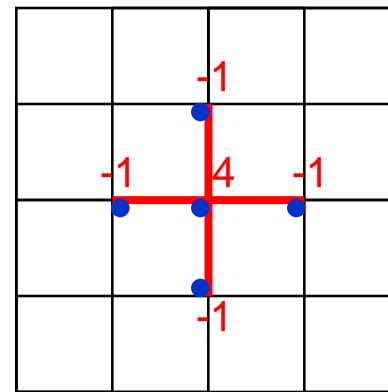
$$L_1 = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}$$



- 2D Poisson equation: solve $L_2 x = b$ where

$$L_2 = \begin{pmatrix} 4 & -1 & & -1 & & & & & & & \\ -1 & 4 & -1 & & & & & & & & \\ & -1 & 4 & & & & & & & & \\ \hline -1 & & & 4 & -1 & & & & & & \\ & -1 & & -1 & 4 & -1 & & & & & \\ & & -1 & & -1 & 4 & & & & & \\ \hline & & & -1 & & & 4 & -1 & & & \\ & & & & -1 & & -1 & 4 & -1 & & \\ & & & & & -1 & & -1 & 4 \end{pmatrix}$$

Graph and "5 point stencil"



3D case is analogous (7 point stencil)

Solving 2D Poisson Equation with FFT (2/2)

- Use facts that:

- $L_1 = FDF^T$ is eigenvalue/eigenvector decomposition, where

- F is very similar to FFT (imaginary part)

$$F(j, k) = \left(\frac{2}{n+1} \right)^{1/2} \cdot \sin \left(\frac{jk\pi}{n+1} \right)$$

- D = diagonal matrix of eigenvalues

$$D(j, j) = 2 \left(1 - \cos \left(\frac{j\pi}{n+1} \right) \right)$$

- 2D Poisson same as solving $L_1 X + X L_1 = B$ where

- X square matrix of unknowns at each grid point, B square too

Solving 2D Poisson Equation with FFT (2/2)

Substitute $L_1 = FDF^T$ into 2D Poisson to get algorithm:

1. Perform 2D "FFT" on B to get $B' = F^T B F$ or $B = F B' F^T$

Get

$$\begin{aligned}(FDF^T)X + X(FDF^T) &= FB'F^T \\ F(D(F^T X F) + (F^T X F)D)F^T &= FB'F^T \\ DX' + X'D &= B'\end{aligned}$$

2. Solve $DX' + X'D = B'$ for X' : $X'(j, k) = B'(j, k) / (D(j, j) + D(k, k))$

3. Perform inverse 2D "FFT" on $X' = F^T X F$ to get $X = F X' F^T$

Cost = 2 2D-FFTs plus n^2 adds, divisions = $O(n^2 \log n)$

Algorithms for 2D Poisson Equation ($N = n^2$ vars)

Algorithm	Serial	PRAM	Memory	#Procs
• Dense LU	N^3	N	N^2	N^2
• Band LU	$N^2 (N^{7/3})$	N	$N^{3/2} (N^{5/3})$	$N (N^{4/3})$
• Jacobi N^2	$(N^{5/3})$	$N (N^{2/3})$	N	N
• Explicit Inv.	N^2	$\log N$	N^2	N^2
• Red/Black SOR	$N^{3/2} (N^{4/3})$	$N^{1/2} (N^{1/3})$	N	N
• Sparse LU	$N^{3/2} (N^2)$	$N^{1/2}$	$N \cdot \log N (N^{4/3})$	N
→ • FFT	$N \cdot \log N$	$\log N$	N	N
• Multigrid	N	$\log^2 N$	N	N
• Lower bound	N	$\log N$	N	

Recall: PRAM is an idealized parallel model with zero cost communication

Reference: table from [Demmel, Applied Numerical Linear Algebra, SIAM, 1997]

Related Transforms

- Most applications require multiplication by both F and F^{-1}

$$F(j, k) = e^{(2\pi ijk/m)}$$

- Multiplying by F and F^{-1} are essentially the same.

$$F^{-1} = \text{complex_conjugate}(F) / m$$

- For solving the Poisson equation and various other applications, we use variations on the FFT
 - The sin transform -- imaginary part of F
 - The cos transform -- real part of F
- Algorithms are similar, so we will focus on F

Serial Algorithm for the FFT

Compute the FFT ($F * v$) of an m -element vector v

$$\begin{aligned}(F * v)[j] &= \sum_{k=0}^{m-1} F(j, k) * v(k) \\ &= \sum_{k=0}^{m-1} \bar{\omega}^{(j*k)} * v(k) \\ &= \sum_{k=0}^{m-1} (\bar{\omega}^j)^k * v(k) \\ &= V(\bar{\omega}^j)\end{aligned}$$

where V is defined as the polynomial

$$V(x) = \sum_{k=0}^{m-1} x^k * v(k)$$

Divide and Conquer FFT

- V can be evaluated using divide-and-conquer

$$\begin{aligned}V(x) &= \sum_{k=0}^{m-1} x^k * v(k) \\&= v[0] + x^2 v[2] + x^4 v[4] + \dots \\&\quad + x(v[1] + x^2 v[3] + v[5] + \dots) \\&= V_{even}(x^2) + x V_{odd}(x^2)\end{aligned}$$

- V has degree $m - 1$, so V_{even} and V_{odd} are polynomials of degree $m/2 - 1$
- We evaluate these at m points: $(\bar{\omega}^j)^2$ for $0 \leq j \leq m - 1$
- But this is really just $m/2$ different points, since

$$(\bar{\omega}^{(j+m/2)})^2 = (\bar{\omega}^j * \bar{\omega}^{m/2})^2 = \bar{\omega}^{2j} * \bar{\omega}^m = (\bar{\omega}^j)^2$$

- So FFT on m points reduced to 2 FFTs on $m/2$ points
 - Divide and conquer!

Divide-and-Conquer FFT (D&C FFT)

FFT($v, \bar{\omega}, m$) ... assume m is a power of 2

if $m = 1$ return $v[0]$

else

$$V_{even} = \text{FFT}(v[0:2:m-2], \bar{\omega}^2, m/2)$$

$$V_{odd} = \text{FFT}(v[1:2:m-1], \bar{\omega}^2, m/2)$$

$$\bar{\omega}_{vec} = [\bar{\omega}^0, \bar{\omega}^1, \dots, \bar{\omega}^{(m/2-1)}] \leftarrow \text{precomputed}$$

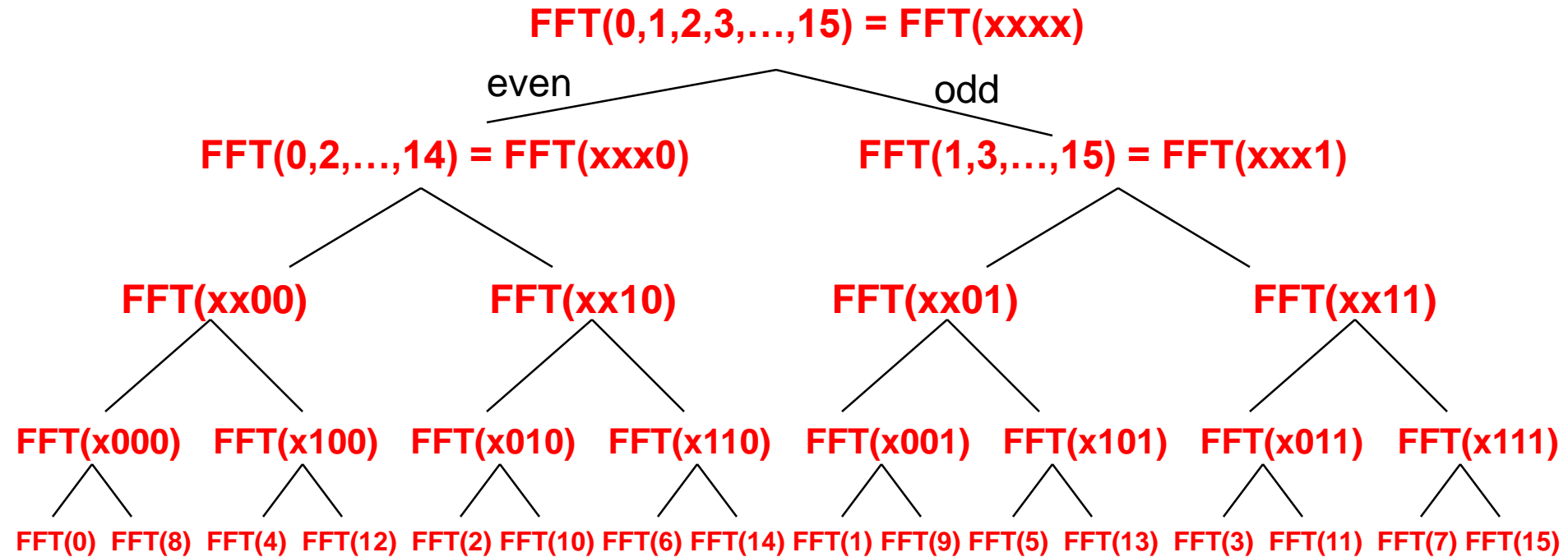
$$\text{return } [V_{even} + (\bar{\omega}_{vec} .* V_{odd}), V_{even} - (\bar{\omega}_{vec} .* V_{odd})]$$

- Matlab notation: " $.*$ " means component-wise multiply.

Cost: $T(m) = 2T(m/2) + O(m) = O(m \log m)$ operations.

An Iterative Algorithm

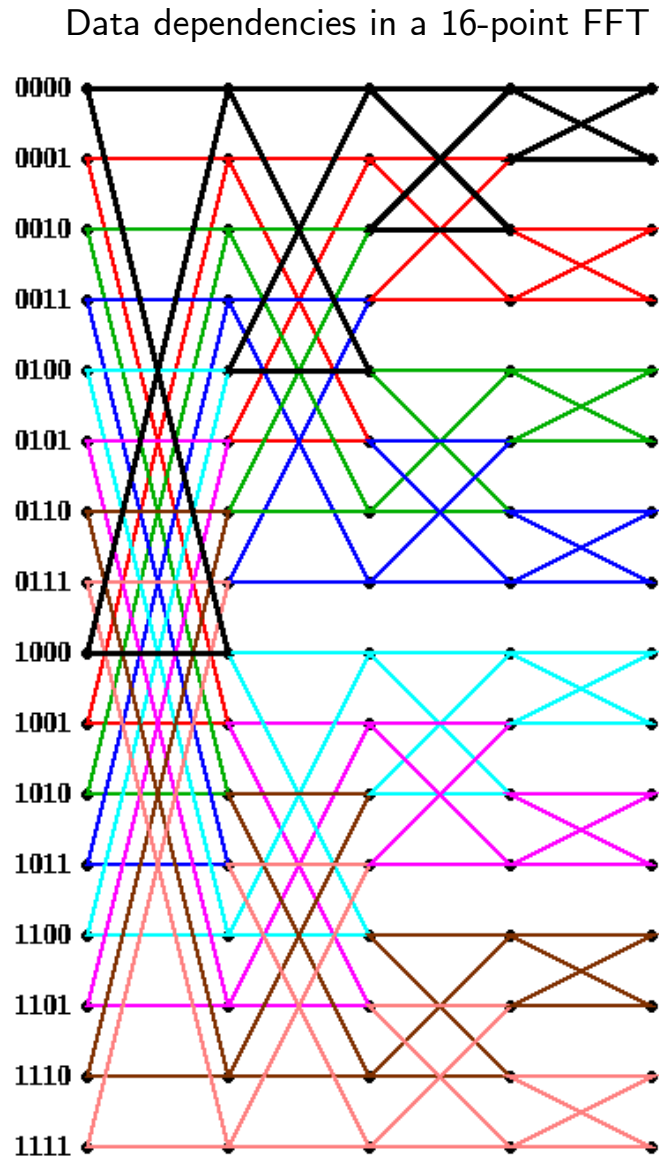
- The call tree of the D&C FFT algorithm is a complete binary tree of $\log m$ levels



- An **iterative** algorithm that uses loops rather than **recursion**, does each level in the tree starting at the bottom
 - Algorithm overwrites $v[i]$ by $(F*v)[\text{bitreverse}(i)]$
- Practical algorithms combine recursion (for memory hierarchy) and iteration (to avoid function call overhead) – more later

Parallel 1D FFT

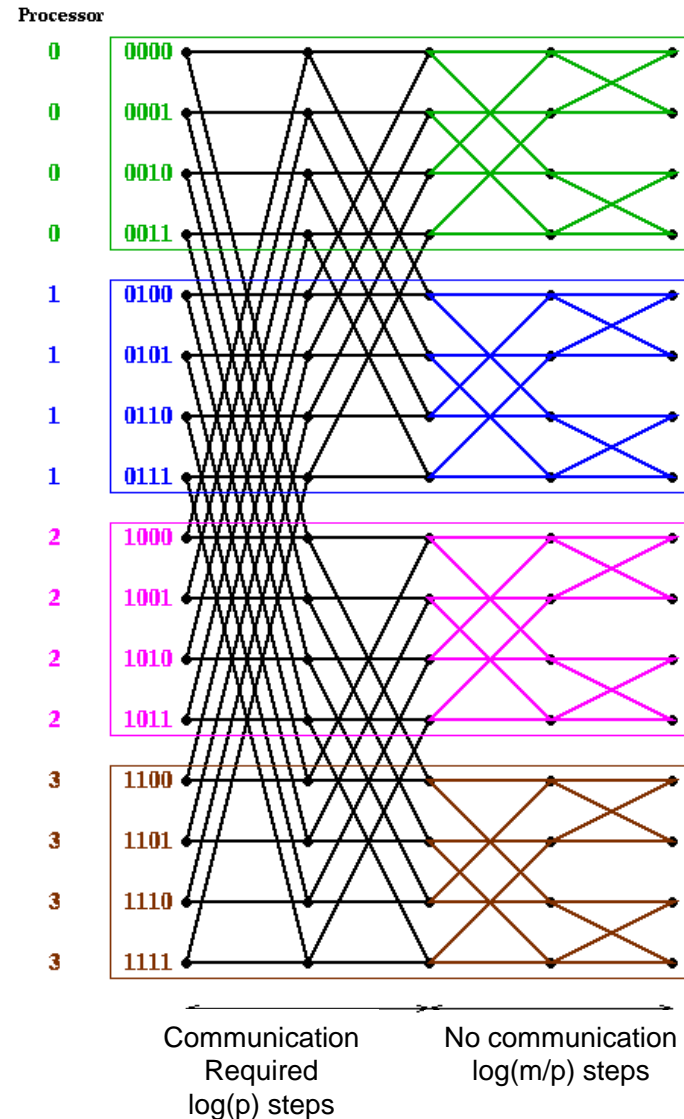
- Data dependencies in 1D FFT
 - Butterfly pattern
 - From $V_{even} \pm \bar{\omega}_{vec} .* V_{odd}$
- A PRAM algorithm takes $O(\log m)$ time
 - each step to right is parallel
 - there are $\log m$ steps
- What about communication cost?



Block Layout of 1D FFT

- Using a block layout (m/p contiguous words per processor)
- No communication in last $\log m/p$ steps
- Significant communication in first $\log p$ steps

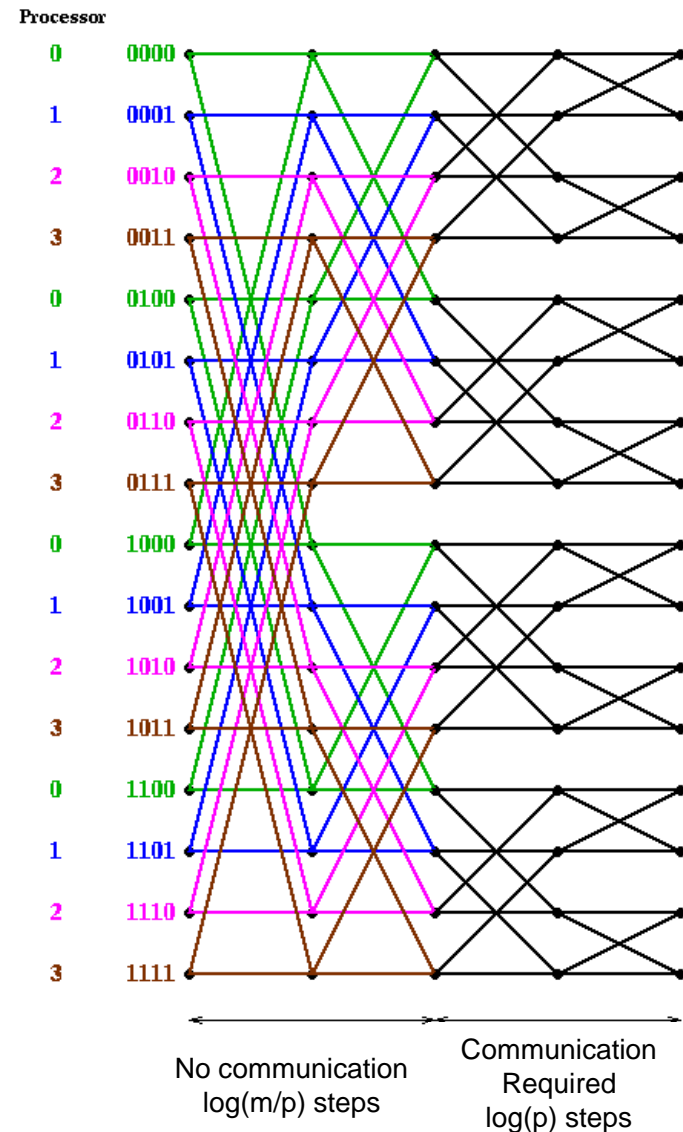
Block Data Layout of an $m=16$ -point FFT on $p=4$ Processors



Cyclic Layout of 1D FFT

- Cyclic layout (consecutive words map to consecutive processors)
- No communication in first $\log(m/p)$ steps
- Communication in last $\log(p)$ steps

Cyclic Data Layout of an $m=16$ -point FFT on $p=4$ Processors

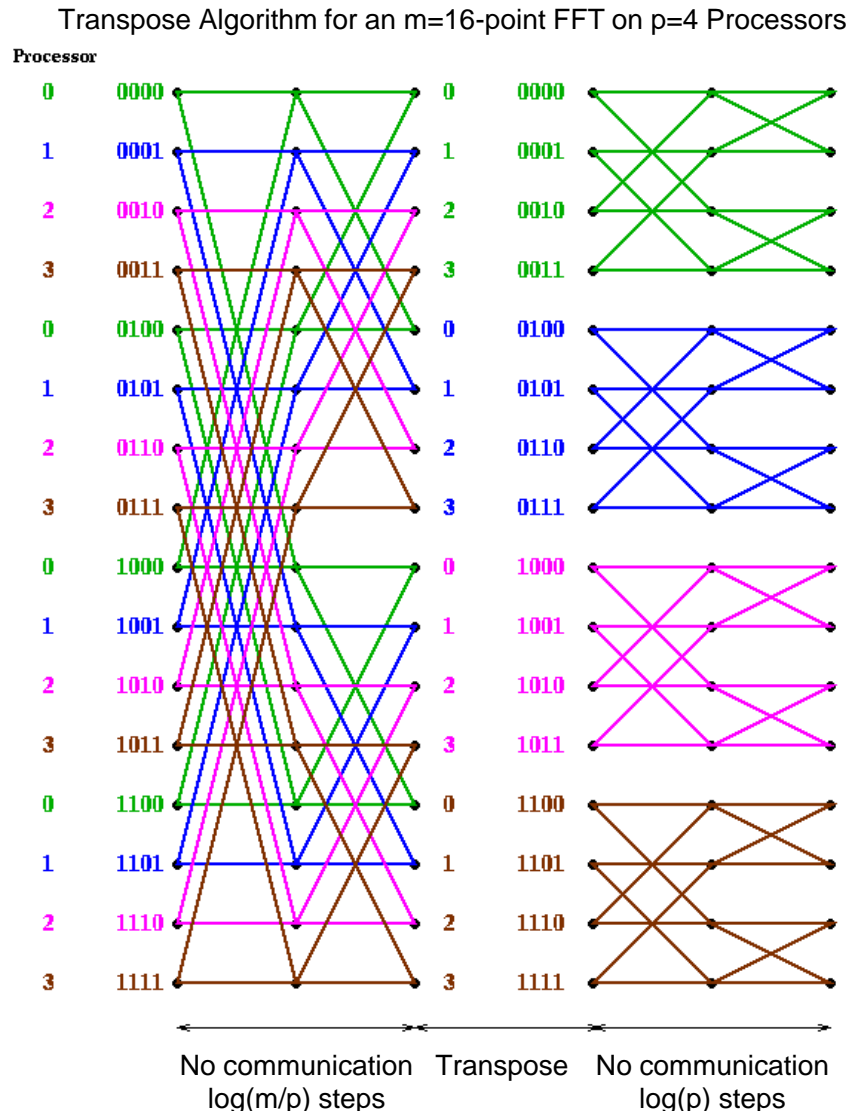


Parallel Complexity

- m = vector size, p = number of processors
- f = time per flop = 1
- α = latency for message
- β = time per word in a message
- $\text{Time}(\text{block_FFT}) = \text{Time}(\text{cyclic_FFT}) =$
 - $2 * m * \log(m) / p$... perfectly parallel flops
 - $+ \log(p) * \alpha$... 1 message/stage, $\log p$ stages
 - $+ (m/p) * \log(p) * \beta$... m/p words/message

FFT With "Transpose"

- If we start with a cyclic layout for first $\log(m/p)$ steps, there is no communication
- Then **transpose** the vector for last $\log(p)$ steps
- All communication is in the transpose
- Note: This example has $\log(m/p) = \log(p)$
 - If $\log(m/p) < \log(p)$ more phases/layouts will be needed
 - We will assume $\log(m/p) \geq \log(p)$ for simplicity



Why is the Communication Step Called a Transpose?

- Analogous to transposing an array
- View as a 2D array of m/p by p

Block Layout

Processor			
0	1	2	3
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Cyclic Layout

Processor			
0	1	2	3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Parallel Complexity of the FFT with Transpose

- Assume no communication is pipelined (overestimate!)
- $\text{Time}(\text{transposeFFT}) =$
 - $2 * m * \log(m) / p$ same as before
 - $+ (p-1) * \alpha$ was $\log(p) * \alpha$
 - $+ m * (p-1) / p^2 * \beta$ was $m * \log(p) / p * \beta$
- If communication is pipelined, so we do not pay for $p-1$ messages, the second term becomes simply α , rather than $(p-1) \alpha$
- This is close to optimal. See LogP paper for details.
- See also following papers
 - A. Sahai, "Hiding Communication Costs in Bandwidth Limited FFT"
 - R. Nishtala et al, "Optimizing bandwidth limited problems using one-sided communication"

Sequential Communication Complexity of the FFT

- How many words need to be moved between main memory and cache of size M to do the FFT of size m , where $m > M$?
- Thm (Hong, Kung, 1981): $\#words = \Omega(m \log m / \log M)$
 - Proof follows from each word of data being reusable only $\log M$ times
- Attained by transpose algorithm
 - Sequential algorithm "simulates" parallel algorithm
 - Imagine we have $p = m/M$ processors, so each processor stores and works on $O(M)$ words
 - Each local computation phase in parallel FFT replaced by similar phase working on cache resident data in sequential FFT
 - Each communication phase in parallel FFT replaced by reading/writing data from/to cache in sequential FFT
- Attained by recursive, "cache-oblivious" algorithm

Parallel Communication Complexity of the FFT

- How many words need to be moved between p processors to do the FFT of size m ?
- Thm (Aggarwal, Chandra, Snir, 1990):

$$\# \text{words} = \Omega\left(\frac{m \log m}{p \log(m/p)}\right)$$

- Proof assumes no recomputation
- Holds independent of local memory size (which must exceed m/p)
- Does TransposeFFT attain lower bound?
 - Recall assumption: $\log(m/p) \geq \log(p)$
 - So $2 \geq \log(m) / \log(m/p) \geq 1$
 - So $\# \text{words} = \Omega(m/p)$
 - Attained by transpose algorithm

Comment on the 1D Parallel FFT

- The above algorithm leaves data in bit-reversed order
 - Some applications can use it this way, like Poisson
 - Others require another transpose-like operation
- Other parallel algorithms also exist
 - A very different 1D FFT is due to Edelman
 - Based on the Fast Multipole algorithm
 - Less communication for non-bit-reversed algorithm
 - Approximates FFT

Higher Dimensional FFTs

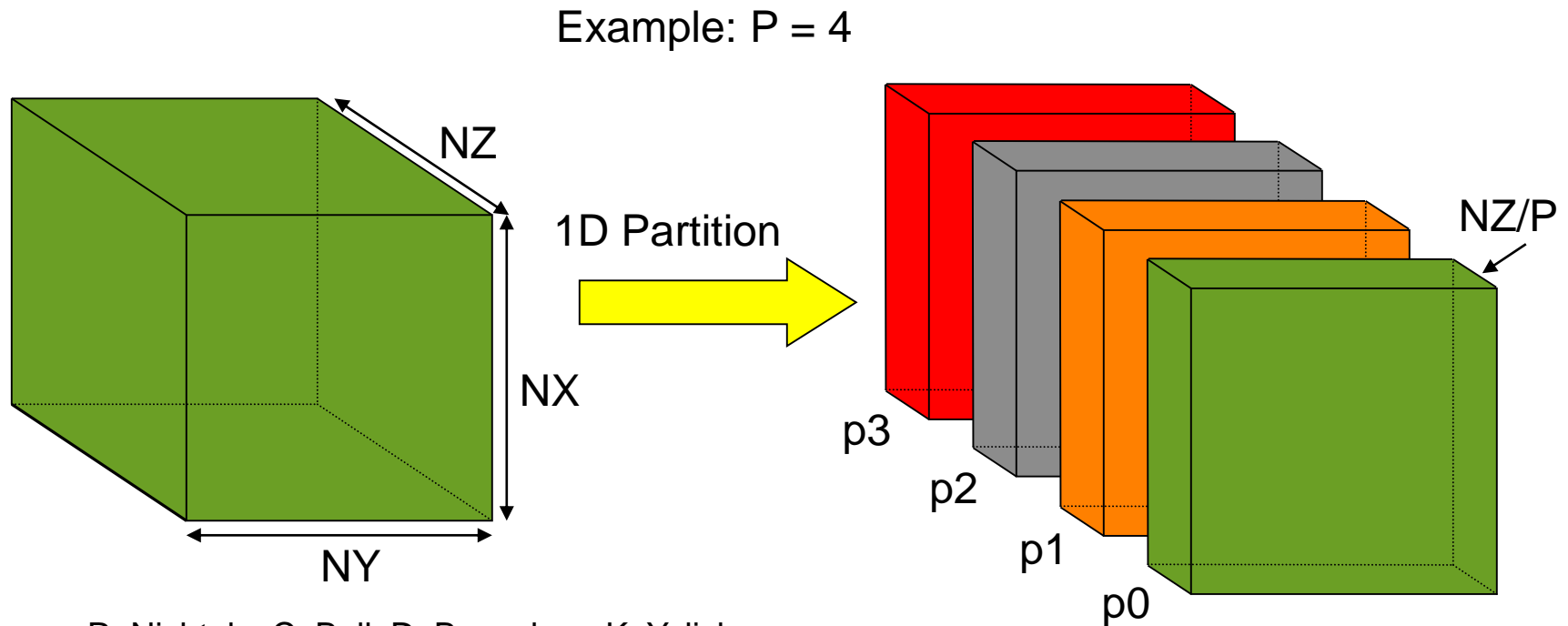
- FFTs on 2 or more dimensions are defined as 1D FFTs on vectors in all dimensions.
 - 2D FFT does 1D FFTs on all rows and then all columns
- There are 3 obvious possibilities for the 2D FFT:
 - (1) 2D blocked layout for matrix, using parallel 1D FFTs for each row and column
 - (2) Block row layout for matrix, using serial 1D FFTs on rows, followed by a transpose, then more serial 1D FFTs
 - (3) Block row layout for matrix, using serial 1D FFTs on rows, followed by parallel 1D FFTs on columns
 - Option 2 is best, if we overlap communication and computation
- For a 3D FFT the options are similar
 - 2 phases done with serial FFTs, followed by a transpose for 3rd
 - can overlap communication with 2nd phase in practice

Bisection Bandwidth

- FFT requires one (or more) transpose operations:
 - Every processor sends $1/p$ -th of its data to each other one
- Bisection Bandwidth limits this performance
 - Bisection bandwidth is the bandwidth across the narrowest part of the network
 - Important in global transpose operations, all-to-all, etc.
- "Full bisection bandwidth" is expensive
 - Fraction of machine cost in the network is increasing
 - Fat-tree and full crossbar topologies may be too expensive
 - Especially on machines with 100K and more processors
 - SMP clusters often limit bandwidth at the node level
- Goal: overlap communication and computation

Performing a 3D FFT (1/3)

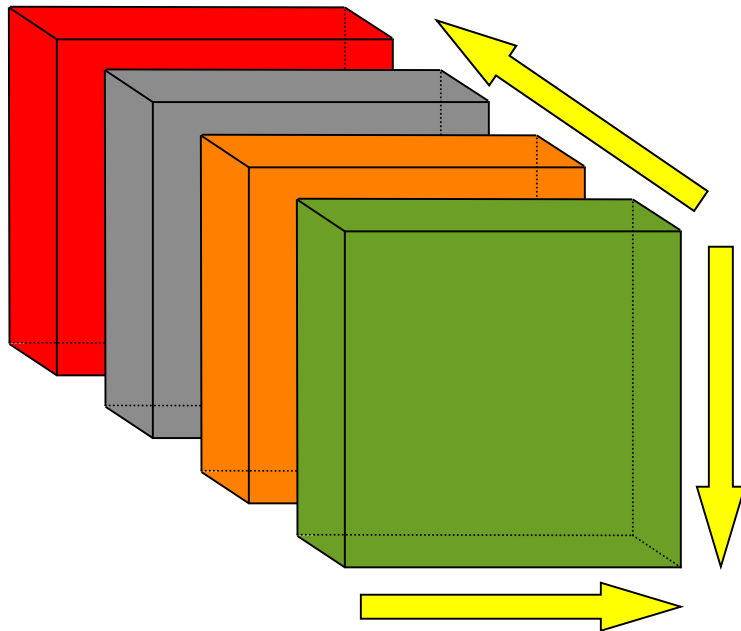
- $NX \times NY \times NZ$ elements spread across P processors
- Will Use 1-Dimensional Layout in Z dimension
 - Each processor gets NZ / P "planes" of $NX \times NY$ elements per plane



Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

Performing a 3D FFT (2/3)

- Perform an FFT in all three dimensions
- With 1D layout, 2 out of the 3 dimensions are local while the last Z dimension is distributed



Step 1: FFTs on the columns
(all elements local)

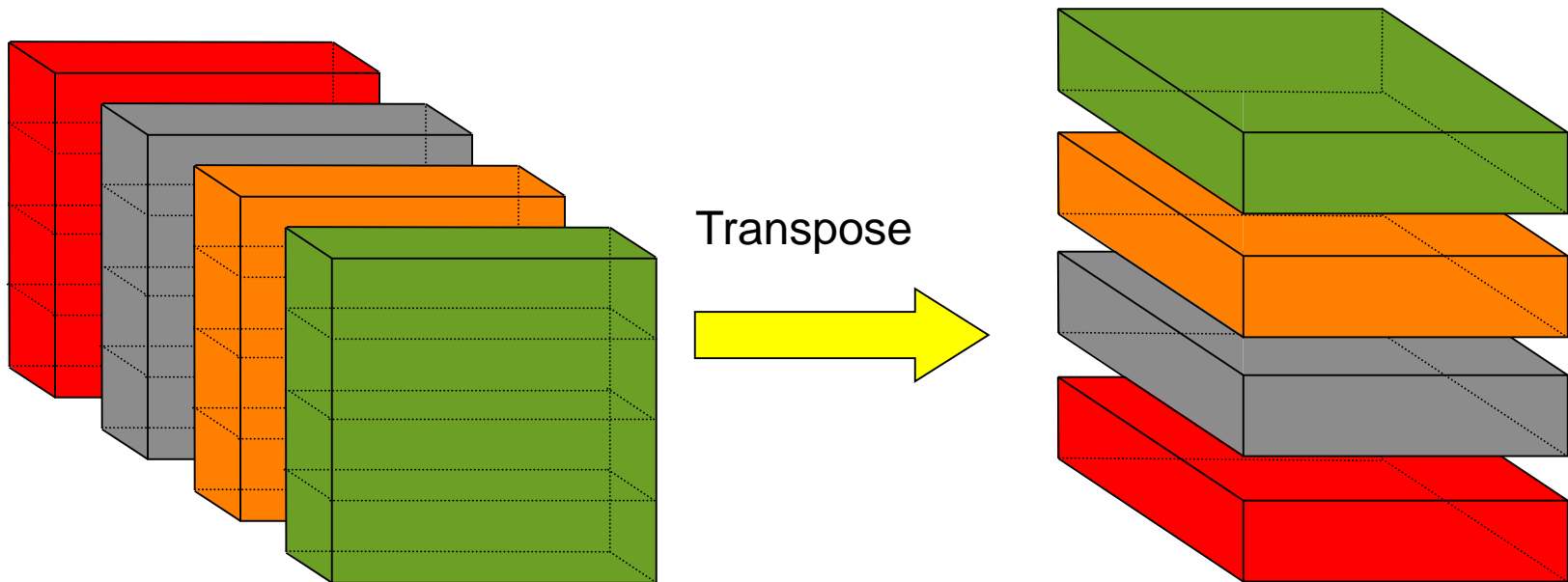
Step 2: FFTs on the rows
(all elements local)

Step 3: FFTs in the Z-dimension
(requires communication)

Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

Performing the 3D FFT (3/3)

- Can perform Steps 1 and 2 since all the data is available without communication
- Perform a Global Transpose of the cube
 - Allows step 3 to continue



Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

The Transpose

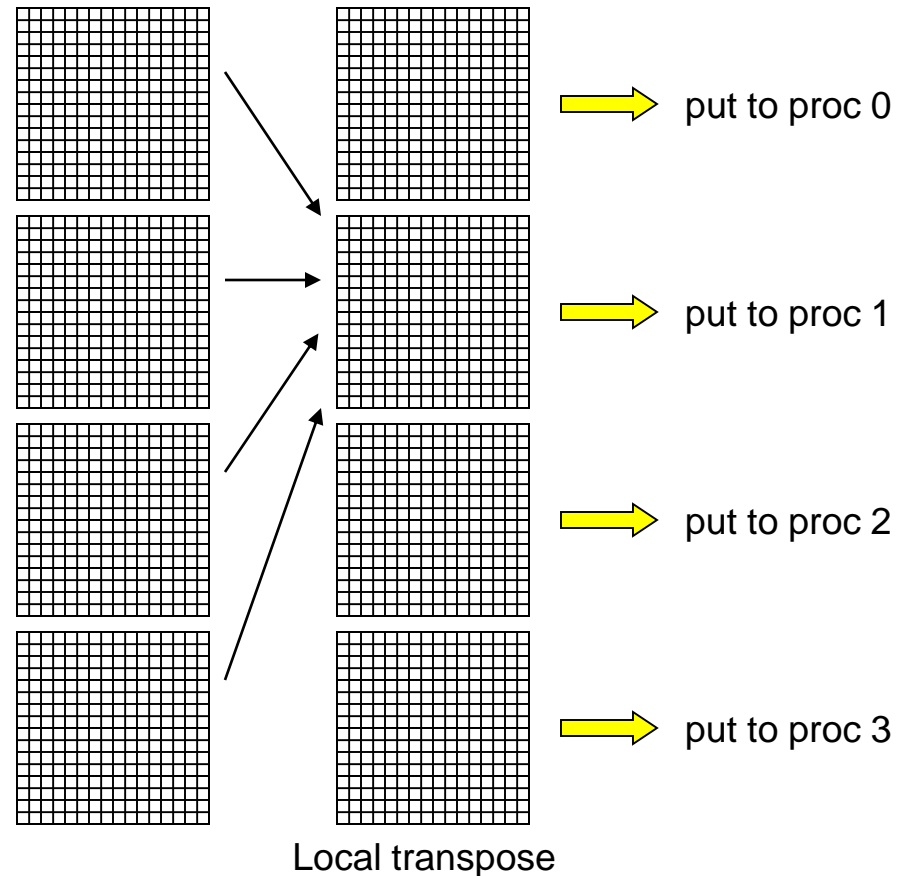
- Each processor has to scatter input domain to other processors
 - Every processor divides its portion of the domain into P pieces
 - Send each of the P pieces to a different processor
- Three different ways to break it up the messages
 1. Packed Slabs (i.e. single packed "All-to-all" in MPI parlance) (3D)
 2. Slabs (2D)
 3. Pencils (1D)
- Going from approach Packed Slabs to Slabs to Pencils leads to
 - An order of magnitude **increase in the number of messages**
 - An order of magnitude **decrease in the size of each message**
- Why do this? Slabs and Pencils allow overlapping communication and computation and leverage RDMA support in modern networks

Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

Algorithm 1: Packed Slabs

Example with $P=4$, $NX=NY=NZ=16$

1. Perform all row and column FFTs
2. Perform local transpose
 - data destined to a remote processor are grouped together
3. Perform P puts of the data



- For 512^3 grid across 64 processors
 - Send 64-1 messages of 512kB each

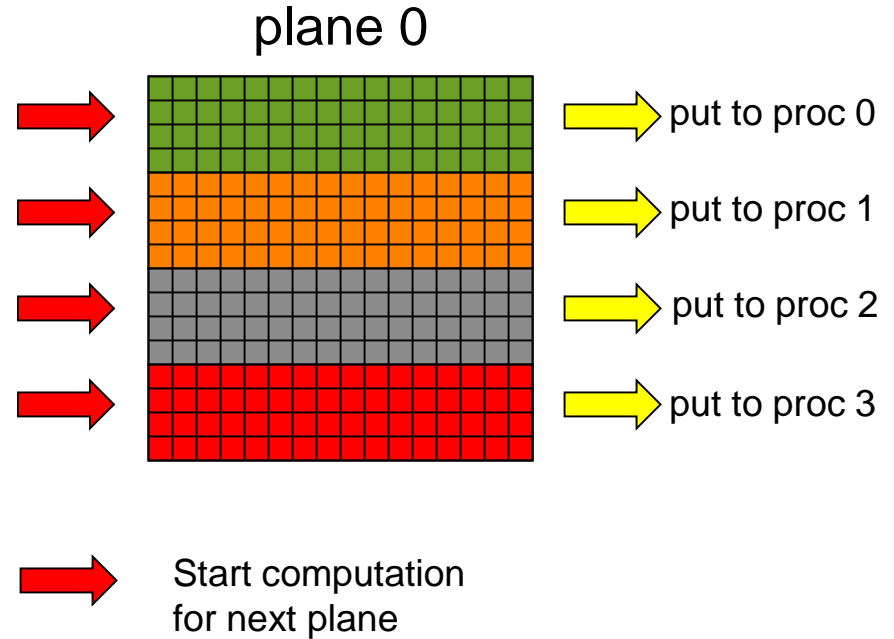
Bandwidth Utilization

- Benchmark: NAS FT with 256 processors on Opteron/InfiniBand
 - Each processor sends 256-1 messages of 512kBytes
 - Global Transpose (i.e. all to all exchange) only achieves 67% of peak point-to-point bidirectional bandwidth
 - Many factors could cause this slowdown
 - Network contention
 - Number of processors with which each processor communicates
- Can we do better?

Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

Algorithm 2: Slabs

- Waiting to send all data in one phase bunches up communication events
- Algorithm Sketch
 - for each of the NZ/P planes
 - Perform all column FFTs
 - for each of the P "slabs"
(a slab is NX/P rows)
 - Perform FFTs on the rows in the slab
 - Initiate 1-sided put of the slab
 - Wait for all puts to finish
 - Barrier
 - Non-blocking RDMA puts allow data movement to be overlapped with computation.
 - Puts are spaced apart by the amount of time to perform FFTs on NX/P rows

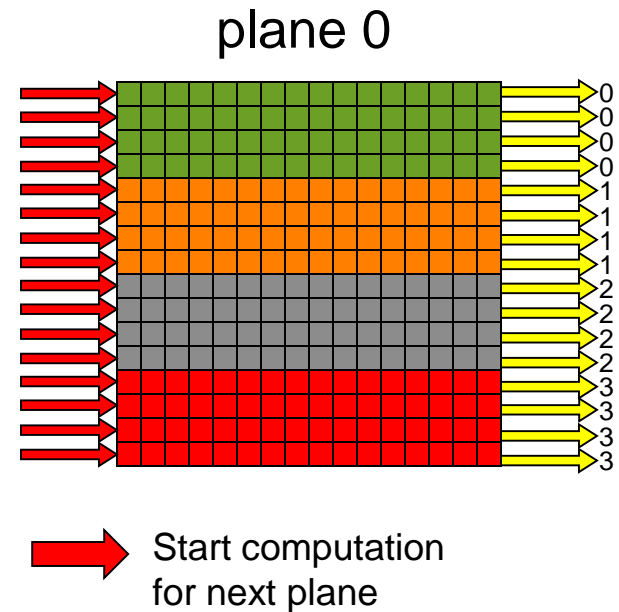


- For 512^3 grid across 64 processors
 - Send 512-8 messages of 64kB each

Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

Algorithm 3: Pencils

- Further reduce the granularity of communication
 - Send a row (*pencil*) as soon as it is ready
- Algorithm Sketch
 - For each of the NZ/P planes
 - Perform all 16 column FFTs
 - For $r=0; r < NX/P; r++$
 - For each slab s in the plane
 - Perform FFT on row r of slab s
 - Initiate 1-sided put of row r
 - Wait for all puts to finish
 - Barrier
- Large increase in message count
- Communication events finely diffused through computation
 - Maximum amount of overlap
 - Communication starts early

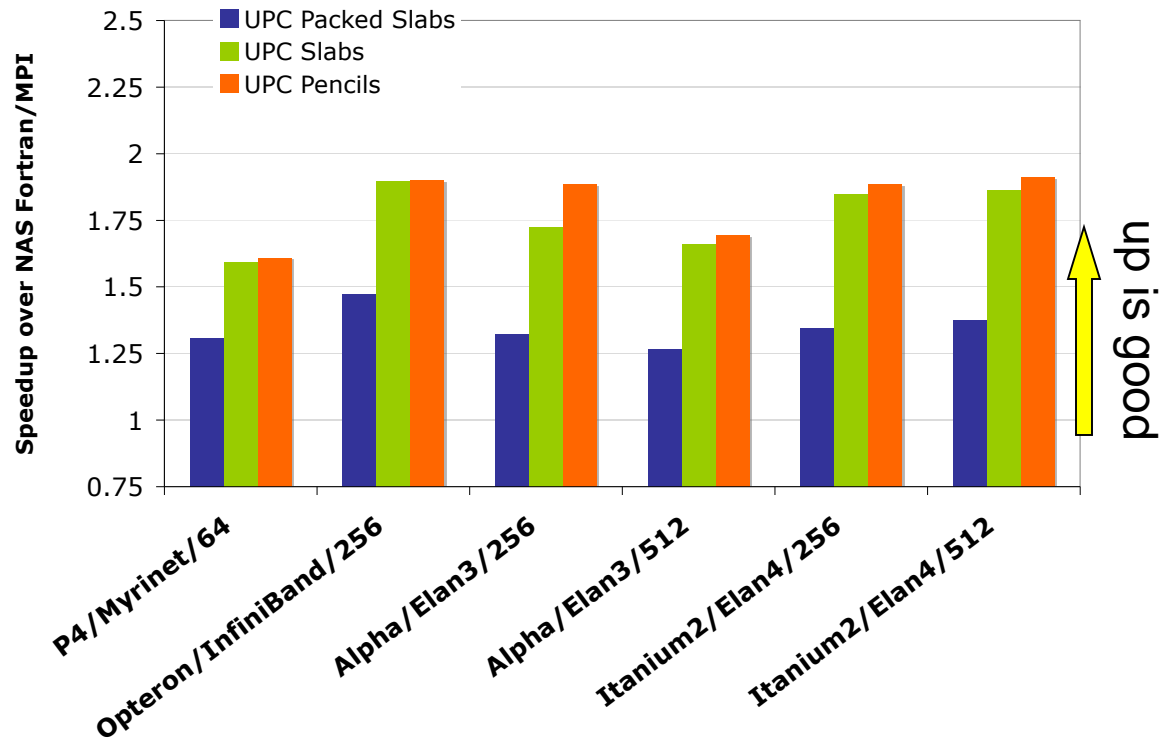


- For 512^3 grid across 64 processors
 - Send $4096-64$ messages of 8kB each

Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

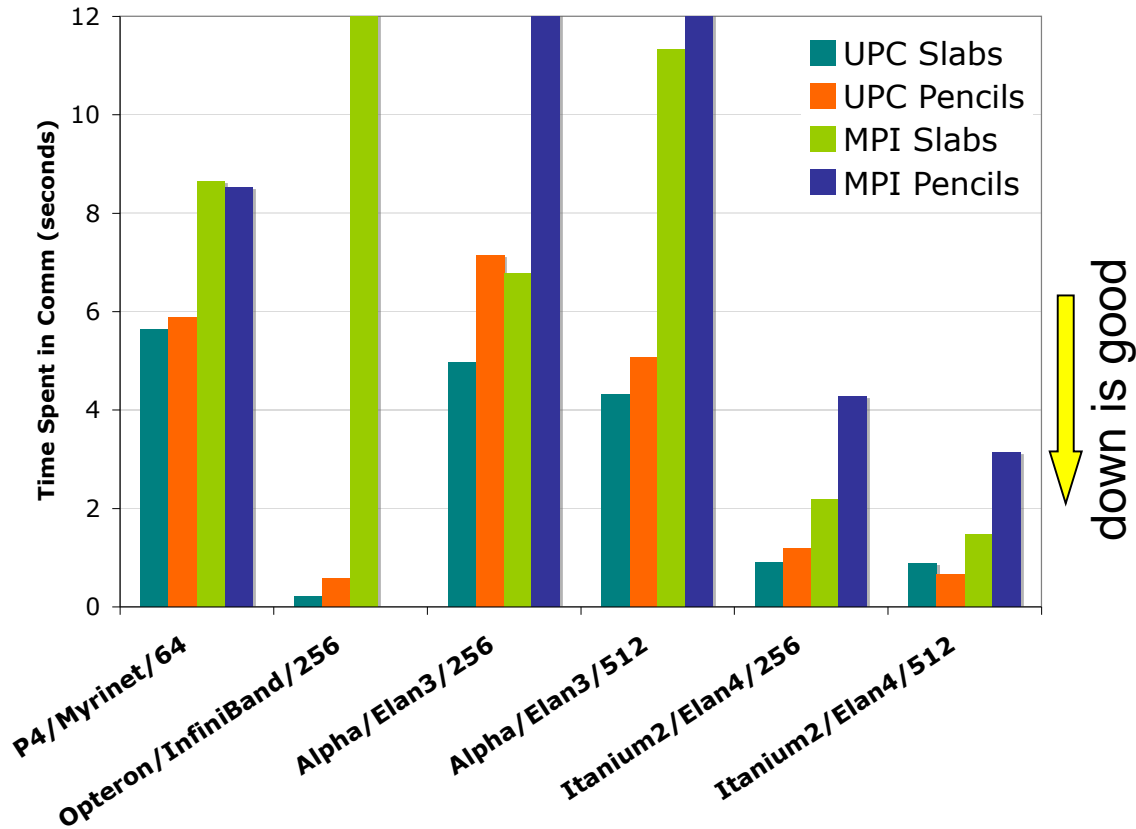
Comparison of Algorithms

- Compare 3 algorithms against original NAS FT
 - All versions including Fortran use FFTW for local 1D FFTs
 - Largest class that fit in the memory (usually class D)
- All UPC flavors outperform original Fortran/MPI implantation by at least 20%
 - One-sided semantics allow even exchange based implementations to improve over MPI implementations
 - Overlap algorithms spread the messages out, easing the bottlenecks
 - ~1.9x speedup in the best case



Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

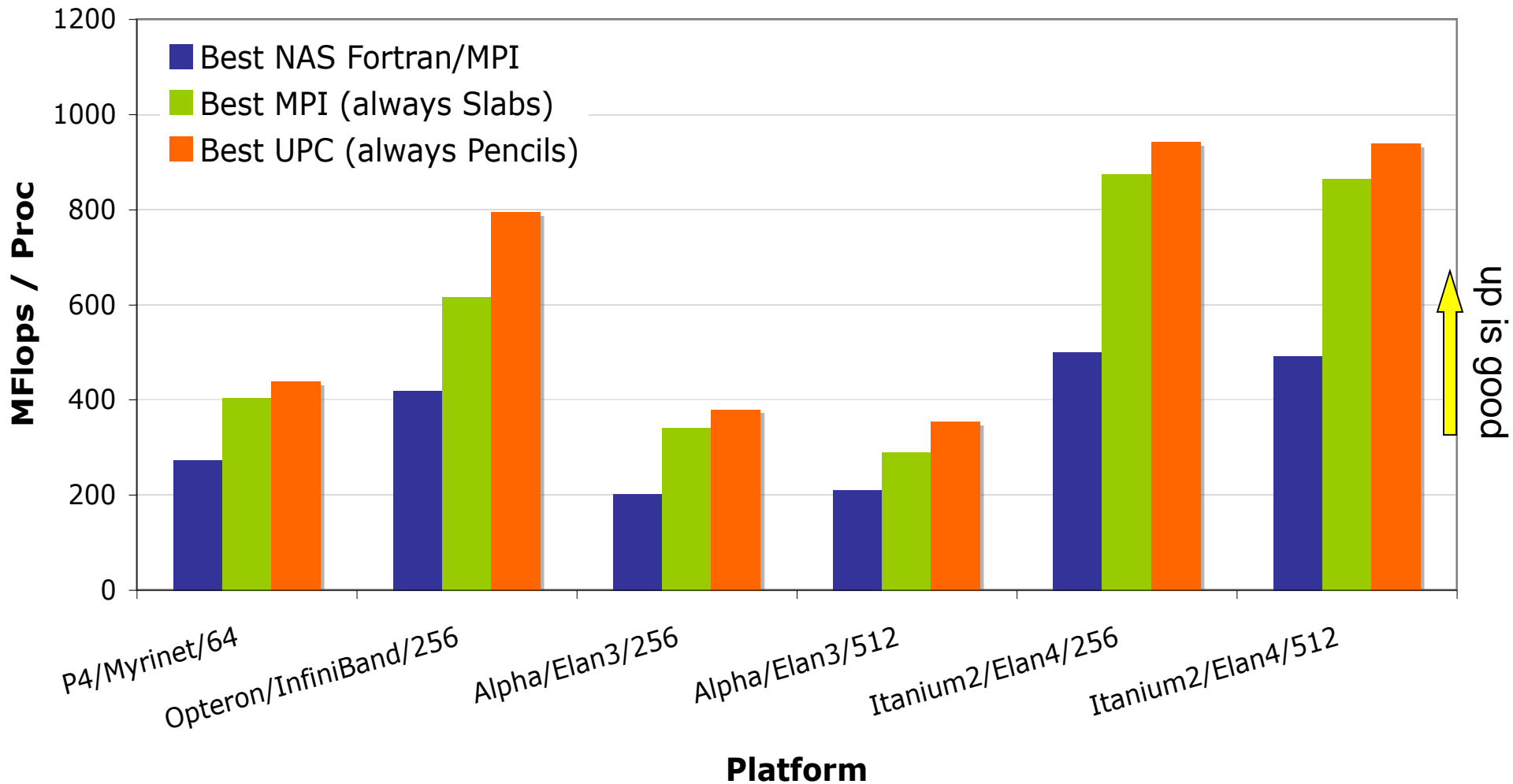
Time Spent in Communication



- Implemented the 3 algorithms in MPI using Irecv and Isends
- Compare time spent initiating or waiting for communication to finish
 - UPC consistently spends less time in communication than its MPI counterpart
 - MPI unable to handle pencils algorithm in some cases

Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

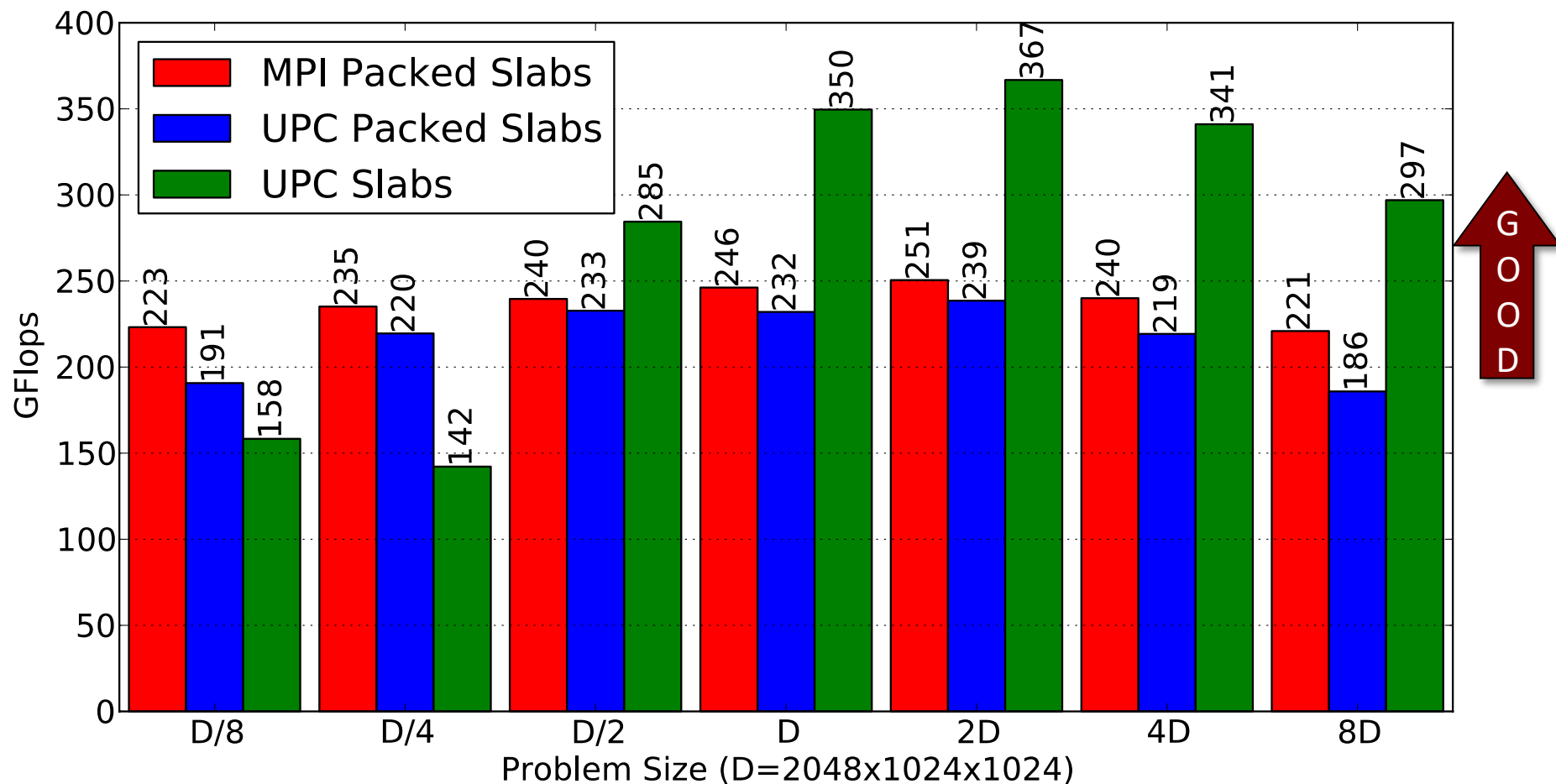
Performance Summary



Source: R. Nishtala, C. Bell, D. Bonachea, K. Yelick

FFT Performance on Cray XT4 (Franklin)

- 1024 Cores of the Cray XT4
 - Uses FFTW for local FFTs
 - Larger the problem size the more effective the overlap



FFTW – Fastest Fourier Transform in the West

- www.fftw.org
- Produces FFT implementation optimized for
 - Your version of FFT (complex, real,...)
 - Your value of n (arbitrary, possibly prime)
 - Your architecture
 - Very good sequential performance
- Won 1999 Wilkinson Prize for Numerical Software
- Widely used
 - Latest version 3.3.8 includes threads, OpenMP
 - Added MPI versions in v3.3
 - supports SSE/SSE2
- GPL license