Trends in Scalable Storage System Design and Implementation

17 April 2012
Prof. Matthew O' Keefe
University of Minnesota
and
Storage Systems Architect, HDS

(Dis)Organization of Talk

- Say’s Law
- Parallel Applications
- Scalable File Systems
  - Posix-Oriented: OrangeFS, Lustre, GPFS,
  - Map-Reduce-Oriented: Google FS, HDFS
  - Relaxed-POSIX: Sorrento, Ceph, Ward Swarms
- Potpourri: Distributed File Systems (NFS, CIFS), Tape, FLASH
- Questions…

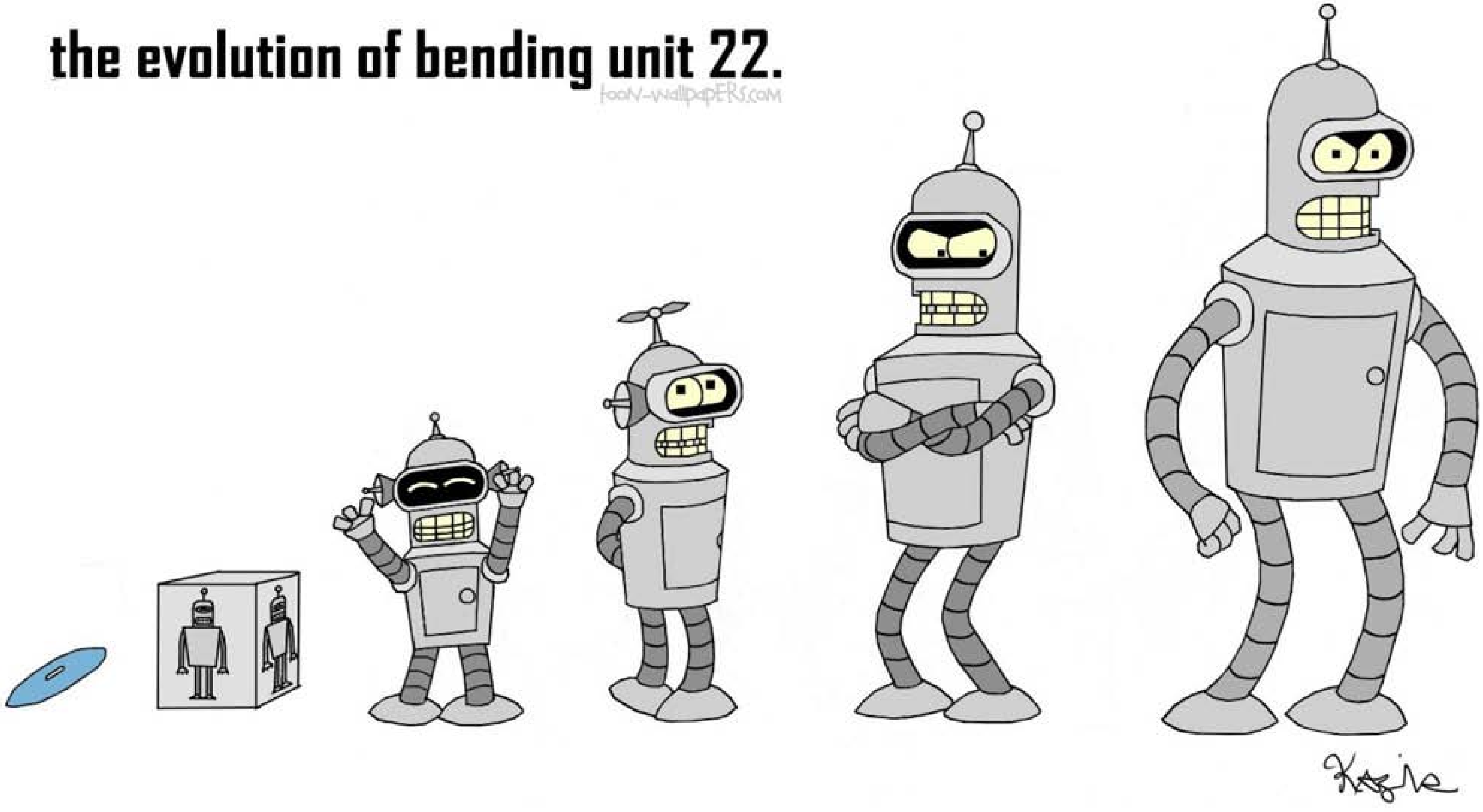
William Faulkner Quote

- “The past is always with us. It isn’t even past.”

Say's Law: (Cheaper) Supply Creates its own Demand (for parallelism)

- Supply on the hardware side:
  - FLASH/NVRAM supplies cheaper IOPS
  - Faster processors, more memory
  - Capacity per drive
  - Bandwidth per drive
  - Network bandwidth
  - Etc.
- Supply on the data side:
  - Clickstream logs
  - Network/server logs
  - Proliferation of IP-enabled sensors
  - Supercomputer output
  - Supercomputer checkpoints
  - Etc.

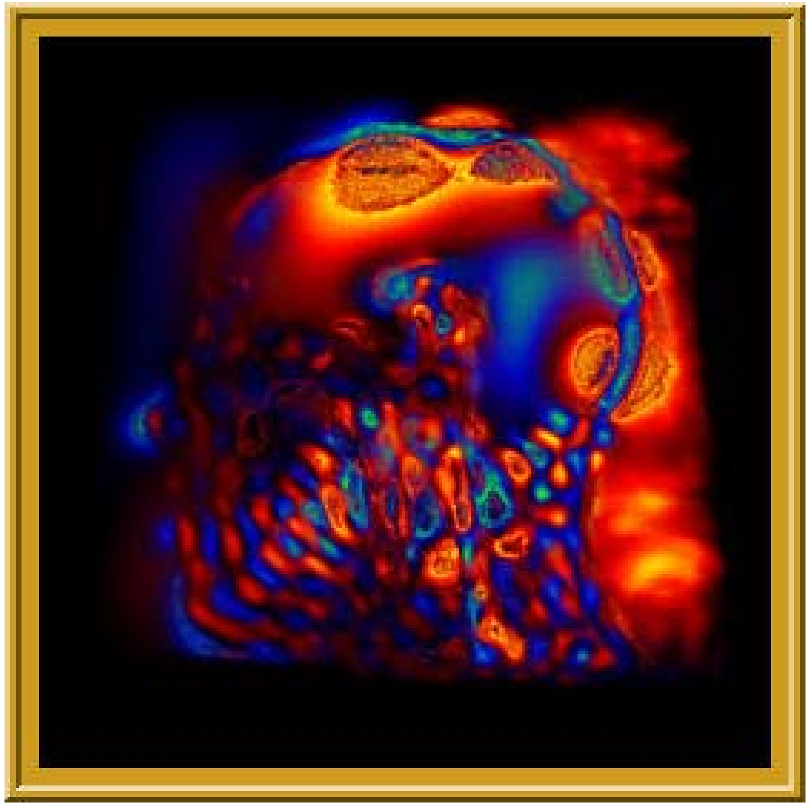
the evolution of bending unit 22.



Building Software for Parallel Systems

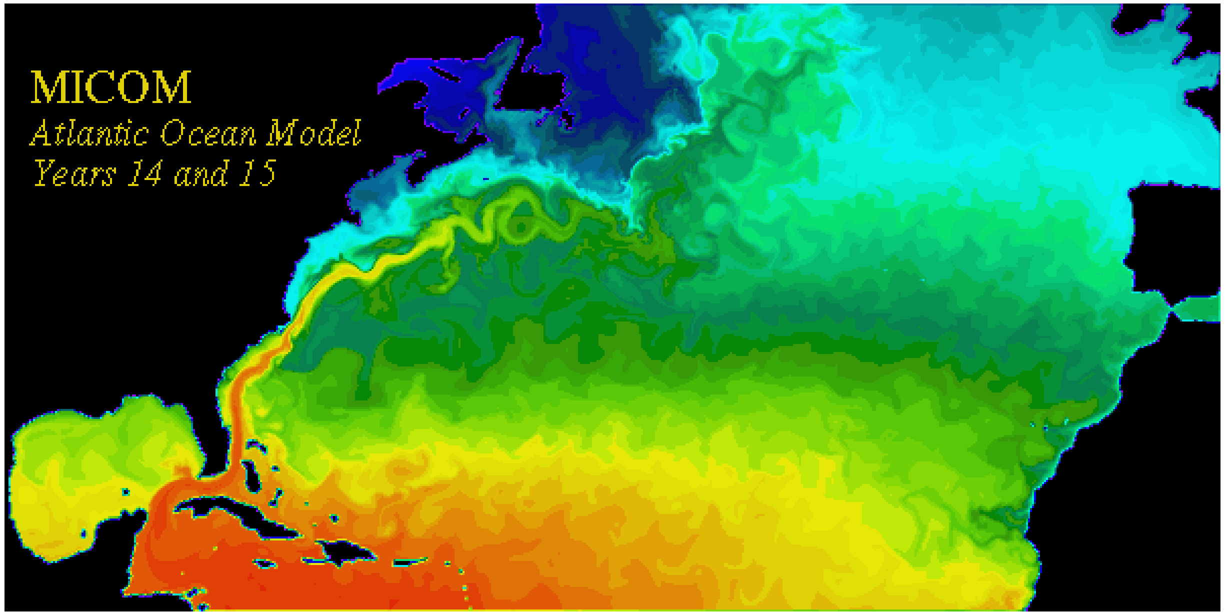
- Detecting parallelism in scientific codes, generating efficient parallel code
- Historically, had been done on loop-by-loop basis
  - Distributed memory parallel computers required more aggressive optimization
  - Parallel programming still a lot like assembly language programming
  - Increasing scope of code to analyze optimize as parallelism increases
  - What's needed is a way to express the problem solution at a much higher level from which efficient code can be generated
- Leverage design patterns and translation technologies to reduce the semantic gap

Parallel Numerical Simulations



- Time Domain Electromagnetics
- Test simulation for parallel electromagnetics code
  - driven by the quest to answer that most pressing of questions?
  - what REALLY happens when you microwave someone's head?
- Magnetic resonance "birdcage" design
- 256x256x256 grid, 2 Gigabytes, 8 processors

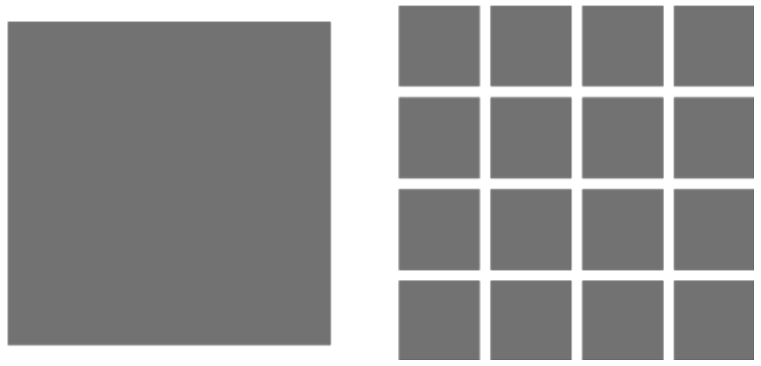
Parallel Numerical Simulations



- Miami Ocean Model — climate simulations extending for centuries
- 1500x1500x11 grid points for North Atlantic: ran on 256 processors of Cray T3D in 1994

Fortran-P Programming Model

- First: don't try to do everything!
- Find the right design pattern: co-design application with parallel system
- Focus on numerical methods that are inherently local and parallel
  - Finite difference, finite volume, high-order compact methods
- The problem should be structured so that the same computations are carried out at each grid point
  - This allows parallelism through simple domain decompositions

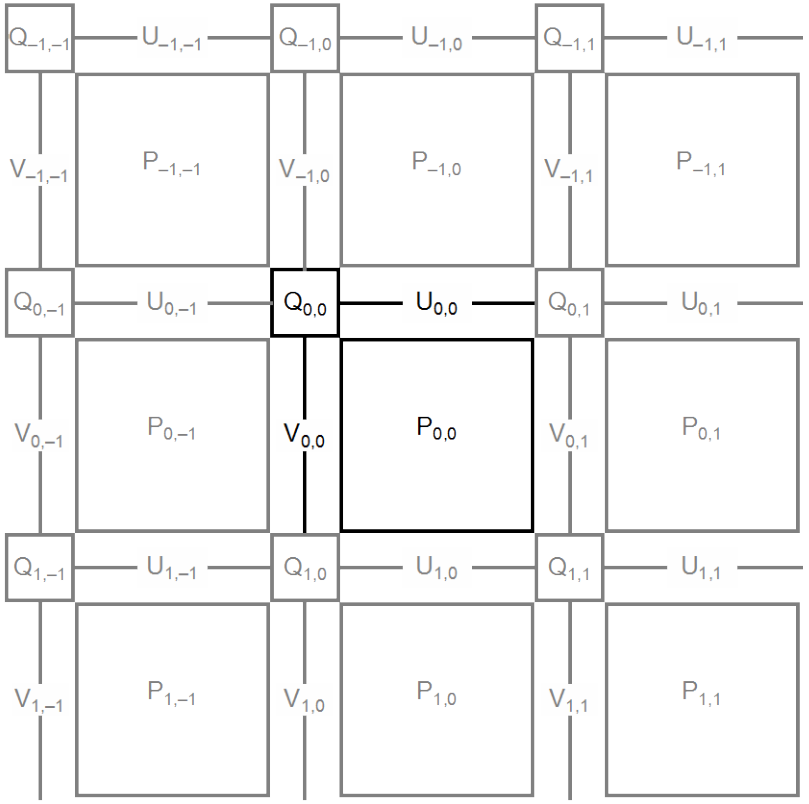


Fortran-P Design Pattern

- Certain loop indices are used to indicate parallel loops
- All loops using those indices are parallel
- The loop indices for the array references are of the form I+a where I is the loop index and a is a small constant (much smaller than the loop bounds)

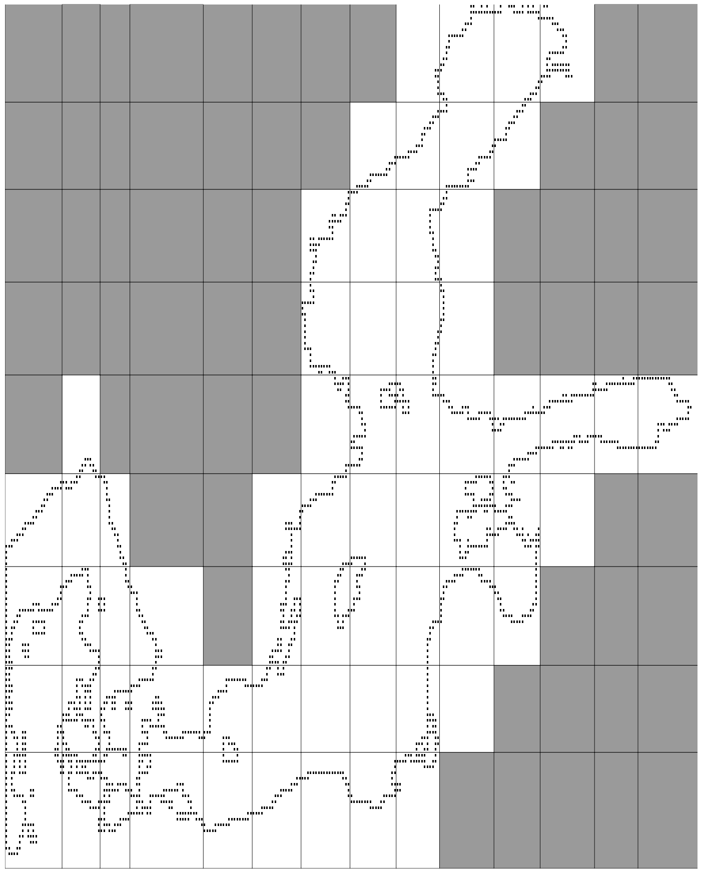
Developing Parallel Numerical Methods

- Higher order compact numerical methods are important to efficient parallel calculations
- These techniques allow fewer grid points to be used compared to traditional centered difference methods (Yee — FDTD)
- PEM — Parallel Electromagnetics Model
  - High-order compact method to solve Maxwell's equations
  - Paul Hayes: the developer
  - Inspired by PPM method of Woodward developed for fluids



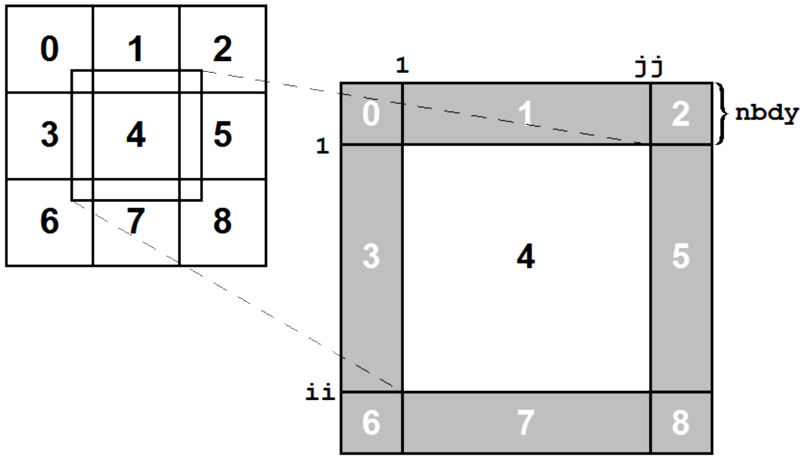
Preparing Input Models: Ocean Circulation

- 3D ocean state at an instant of time: must be constructed from data taken at different times
- Requires sophisticated signal analysis and smoothing
- Some oceanographers spend their whole careers on this problem
- Tools can be developed by re-using existing technology for solid modeling, image processing, and special effects



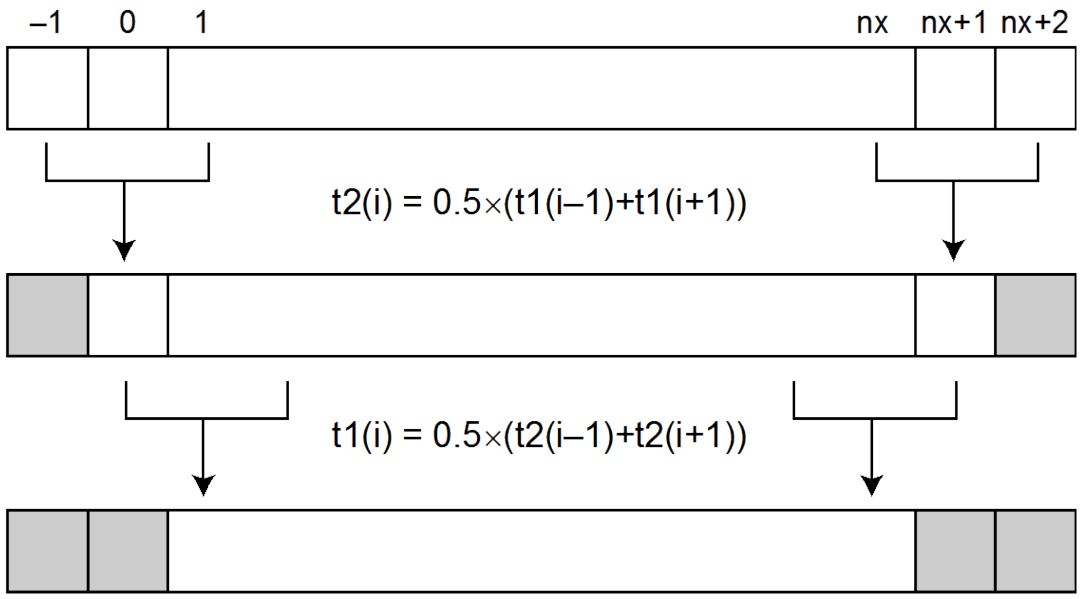
Program □Analysis Tool

- Tool for Parallelism using Additional Zones (TOPAZ)
- Analyzes data flow among parallel arrays: used to extend local computation into neighboring processor' s domain
- Yields large amounts of independent, parallel work for the parallel machine



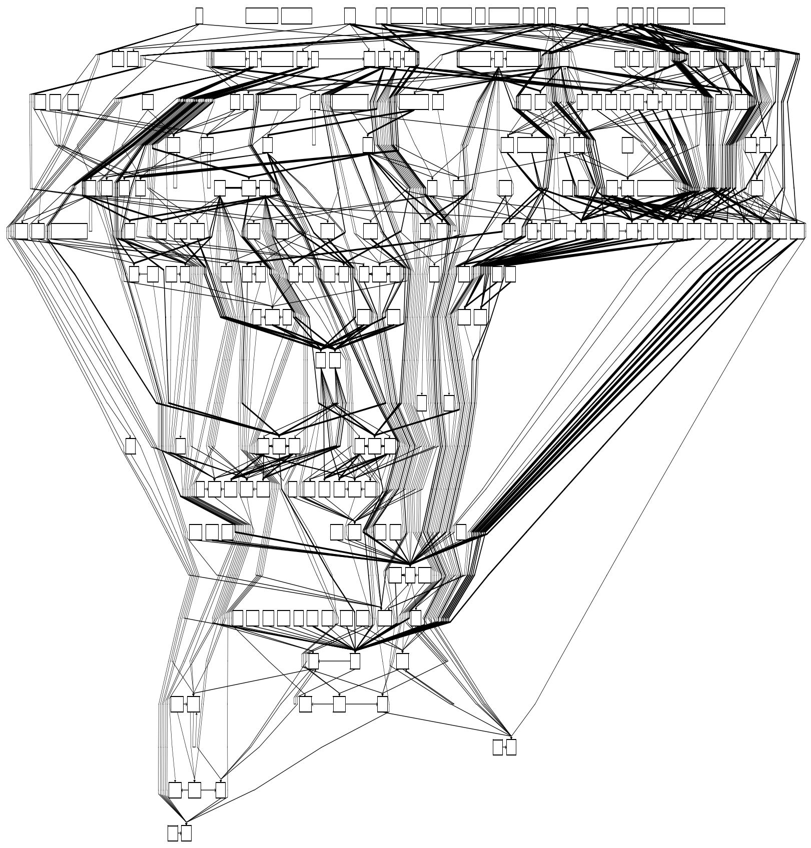
Program Analysis for Fortran-P

- Builds a D-graph from the Static Single Assignment def-use graph of the array flow
- Performed over a "parallel region" of code
- Computes overlaps required by the program to achieve independent parallel computations

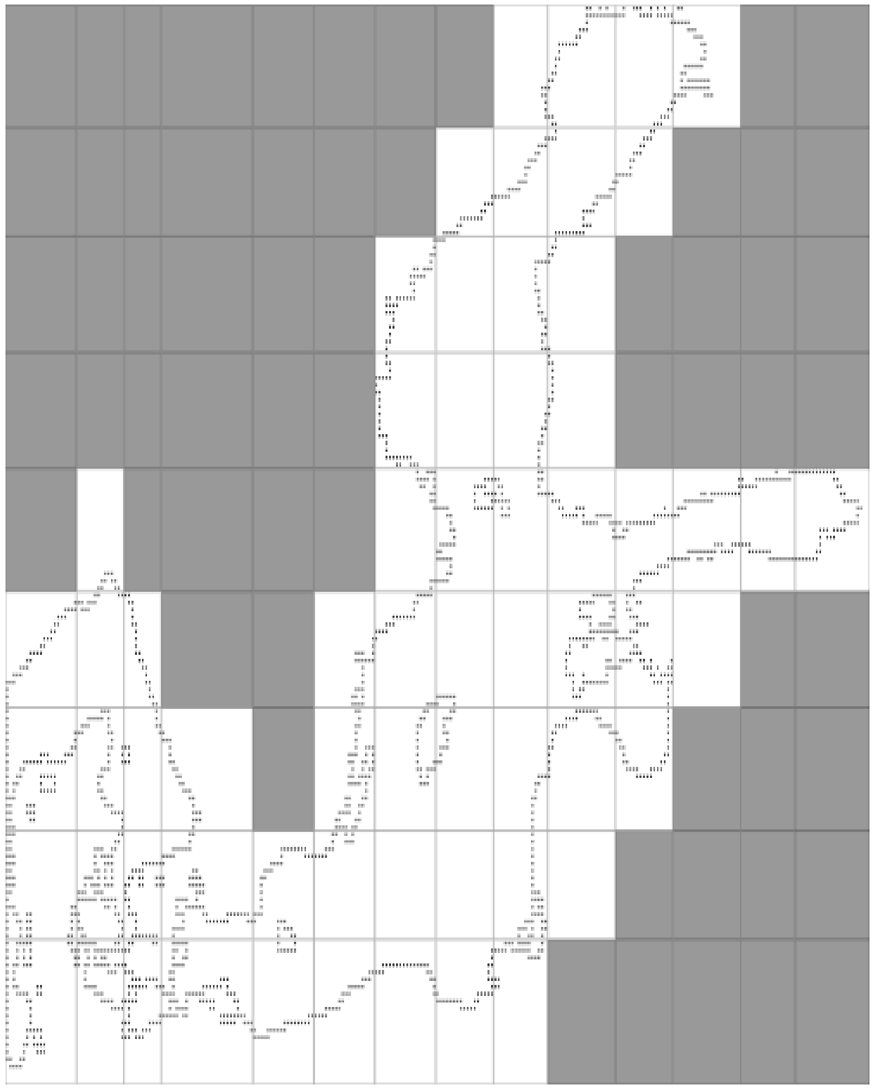


Program □Analysis via D-Graph

- *Offset location*: location where an array reference is accessed
  - x(I-1): -1
- *Offset distance:* difference between the offset location on the LHS and reference on RHS
  - dm(I) = m(I+1) - m(I-1)
  - +1 -1
- *Dependence Range:* sum of offset distances over all possible data flow paths between definition and later reference



Exploiting Problem Structure



Developing Parallel Applications

- For three-dimensional time-dependent solutions, parallelism is required
- *Parallel numerical simulation* involves 4 distinct disciplines

  [1] The science and engineering of the phenomenon simulated

  [2] The mathematics including the numerical methods used

  [3] Software engineering, including code design

  [4] Parallel processing, including systems programming
- Amdahl's Law is a stern taskmaster:

$$Speedup_{overall} = \frac{ExecutionTime_{old}}{ExecutionTime_{new}} = \frac{1}{(1-Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

The Challenge of Amdahl's Law

- Amdahl's Law can be used to determine how much parallelism a given application can usefully exploit
  - let's plug in some numbers to get some intuition about this
- If an application is *99% parallel* and we execute it on a *100-processor machine*, what is the maximum speedup we achieve?

  [a] 99

  [b] 75

  [c] 50
- If an application is *99.9% parallel* and we execute it on a 500-processor machine, what is the maximum speedup achieved?

  [a] 482

  [b] 453

  [c] 333

The Amdahl's Law Challenge

- Fortunately, most applications have tons of parallelism
- Why is that? Because at a small enough time scales all physics is *local*
  - in 1.0 nanosecond, light travels about 0.3 meters
- However, though the physical equations are generally completely parallel
  - there are many ways that we can lose parallelism when we implement the equations in software
  - *let us count the ways...*

The Amdahl's Law Challenge

[1] The numerical method can preclude parallelism

[2] The numerical method may be parallel, but its expression in the actual software may be serial

[3] The compiler may be unable to recognize the parallelism in the software

- poorly written DO loops, aliasing, badly-written code (this is amazingly easy to do)

[4] It may be impossible to express the parallelism in the numerical method in the language

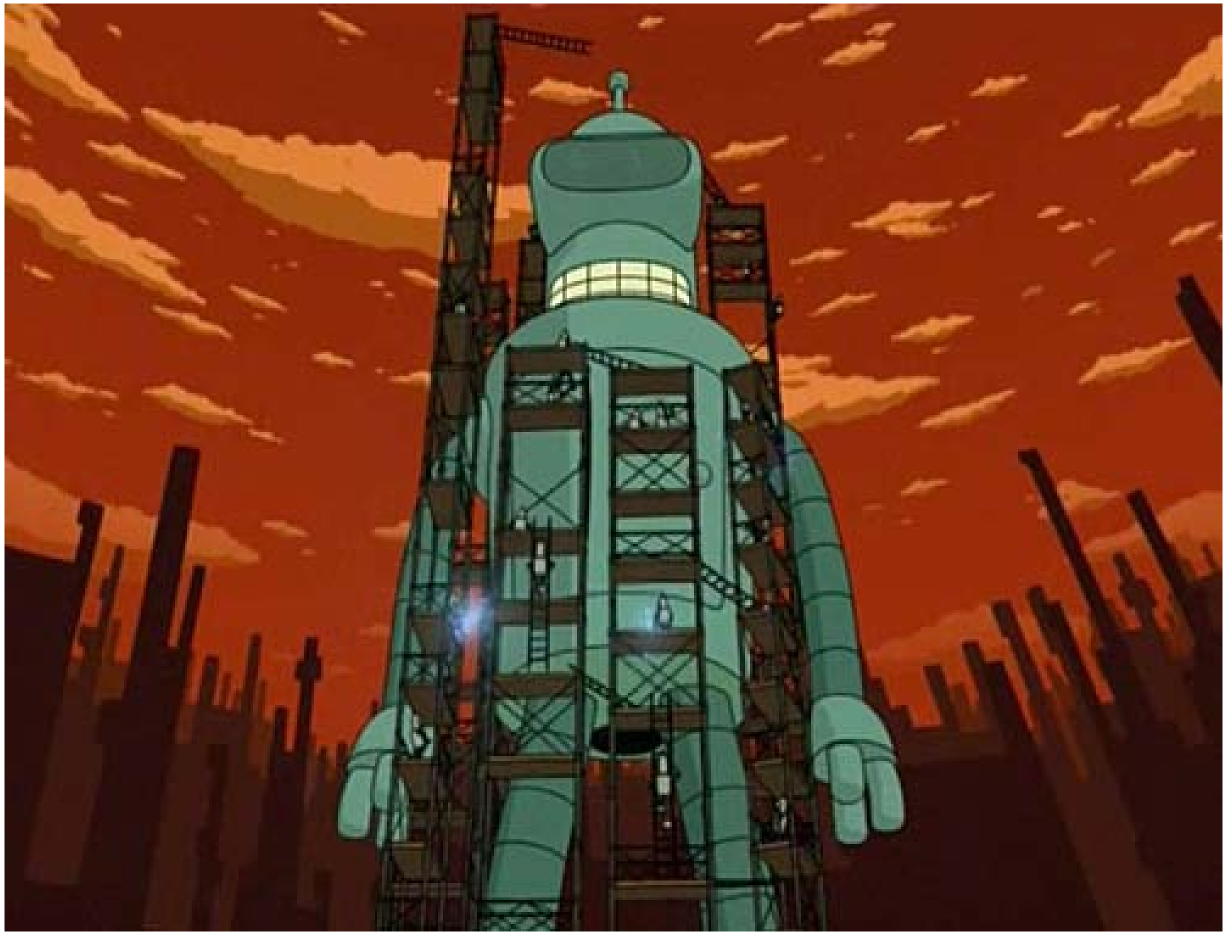
The Amdahl’s Law Challenge

[5] Even if the parallelism is recognized by the compiler or expressed by the programmer in the language, the compiler may do a poor job of mapping the program parallelism to the machine parallelism

[6] There are very subtle effects that can happen during execution even when well written parallel code that is efficiently encoded and mapped to an architecture

- load imbalance
  - due to the application and data
  - due to the machine (network congestion, cache behavior, IO subsystem)
- small data sets and imbalance between computation and communication
- serial bottlenecks
- the OS jitter issue popularized by Sandia

Keeping Pace with Parallel Systems Making Storage Systems Go Faster and Scale More



Storage System Scalability/Speed

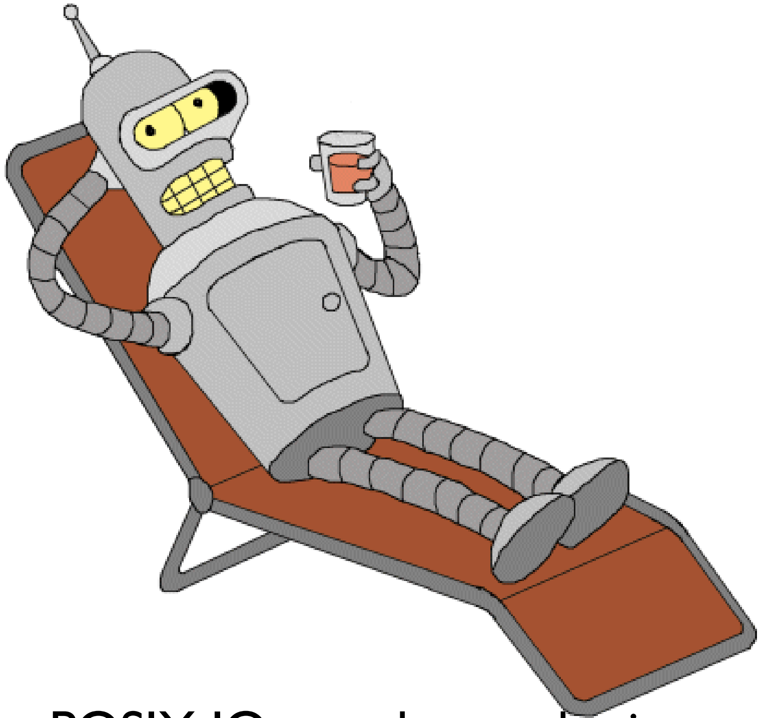
- 1990s: Storage interface standards lacked ability to scale in both speed and connectivity
- Industry responded to this with new standard: Fibre Channel, SATA, etc.
- Allowed shared disks, but system software like file systems and volume managers not built to exploit this
- **Same old story: software catching up with hardware**
- Parallel/cluster file system development begins in the 1990
  - And not just shared disk file systems
- Variety of commercial and open source implementations:
  - All assumed you had to support POSIX or something close to it
  - PVFS, GPFS, GFS, StorNext, CXFS, etc.
- Today, acceleration in hardware technologies continues SSD performance, interface performance, capacities, network performance, etc.

Making Storage Systems Faster and More Scalable

- GFS (Minnesota Global File System) pioneered several interesting techniques for cluster (shared disk) file systems:
  - no central metadata server
  - distributed journals for performance, fast recovery
  - first Distributed Lock Manager for Linux — now used in other cluster projects in Linux
- Implemented POSIX IO
  - Assumption at time was: POSIX is all there is, have to implement that
  - Kind of naïve, assumed POSIX model was the right one
  - UNIX/Windows view of files as linear stream of bytes which can be read/written to anywhere in file by multiple processors
  - Large files, small files, millions of files, directory tree structure, synchronous write/read semantics, etc. all make POSIX difficult to implement

Why POSIX File Systems Are Hard

- They're in the kernel and tightly integrated with complex kernel subsystems like virtual memory
- Byte-granularity, coherency, randomness
- Users expect them to be extremely fast, reliable, and resilient
- Add parallel clients and large storage networks (e.g., Lustre or Panassas) things get even harder



• POSIX IO was the emphasis for parallel HPC IO (1999 through 2010) until recently
• HPC community re-thinking this
• Web/cloud has already moved on

Meanwhile: Google File System and its Clone (Hadoop) use Co-Design

- Google and others (Hadoop) went a different direction: change the interface from POSIX IO to something inherently more scalable
- Users have to write (re-write) applications to exploit the interface
- All about scalability — using commodity server hardware — for a specific kind of workload
- Hardware-software co-design: restricted semantics
  - append-only write semantics from (parallel) producers
  - mostly write-once, read many times by consumers
  - explicit contract on performance expectations: small reads and writes — Fuggedaboutit!
- Very successful, and Hadoop is becoming something of an industry standard
- Lesson: if solving the problem is really, really hard, look at it a different way, move interfaces around, change your assumptions (e.g., as in the parallel programming problem)

Google/Hadoop File Systems

- Google needed a storage system for its web index, various applications — enormous scale
  - GFS paper at FAST conference in 2004 led to development of Hadoop, open source GoogleFS clone
- Co-designed file system with applications
- Applications use map-reduce paradigm
  - Streaming (complete) reads of very large file/datasets, process this data into reduced form (e.g., an index)
  - Files access is write-once, append-only, read-many

Map-Reduce

- cat * | grep | sort | unique -c | cat > file
- input | **map** | shuffle | **reduce** | output
- Simple model for parallel processing
- Natural for:
  - Log processing
  - Web search indexing
  - Ad-hoc queries
- Popular at Facebook, Google, Amazon etc. to determine what ads/products to throw at you
- Hadoop/Map-Reduce starting to replace traditional enterprise data warehouses with low-cost clusters

Scalable File System Goals

- Build with commodity components that frequently fail (to keep things cheap)
  - So design assumes failure is common case
- Nodes incrementally join and leave the cluster
- Scale to 10s to 100s of Petabytes, headed towards exabytes; 1000's to 10s of 10,000s of storage nodes and clients
- Automated administration, simplified recovery (in theory, not practice)

More Scalable Storage Clusters

- <u>Ceph – Sage Weil, UCSC : POSIX lite</u>
  - Multiple metadata servers, dynamic workload balancing
  - Mathematical hash to map file segments to nodes
- <u>Sorrento – UCSB : POSIX with low write-sharing</u>
  - Distributed algorithm for capacity and load balancing, distributed metadata
  - Lazy consistency semantics
- <u>Ward Swarms – Lee Ward, Sandia</u>
  - Similar to Sorrento, uses victim cache and storage tiering, allows parallel writes to any available storage node for performance (like Hadoop)

Open Questions in Scalable Storage

- Tape's role:
  - Accept the fact that its not going away
  - Tape is still the best technology for the providing infinite data capacity
    - Disk file systems do rude things when they run out of space
- Parallel Distributed File System API going Forward
  - pNFS tractions seems limited, but only time will tell
  - SMB 2.x/3.x making performance strides, but not parallel currently
  - POSIX versus Map/Reduce versus ???
- Extreme scalability (e.g., exascale) or Federated Designs
- Ahmdahl's Law and storage systems

Issues in Scaling Parallel File System Workloads

- Disk drive and disk array performance characteristics
  - bit error recovery, vibration tolerance, …
  - basic randomness of seek operation
- File system and operating system software bottlenecks
  - Caching (or not), fs fragmentation, virtual memory randomness, etc.
- Application issues
  - level of parallelism in IO
- Amdahl's Law effects in large-scale parallel IO

Aggregate IO Load Imbalance

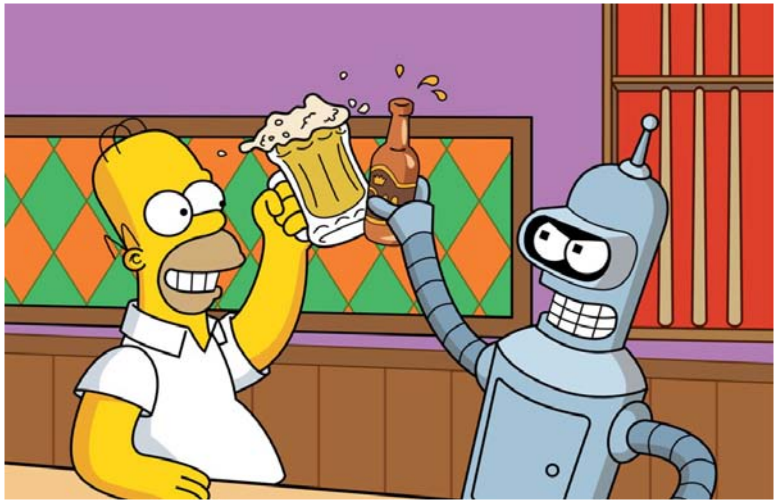
- *LI = (Tmax – Tavg)/Tavg* where *Tmax* is the maximum time to complete an IO request across *n* nodes, and *Tavg* is the average time to complete the IO requests for the *n* nodes

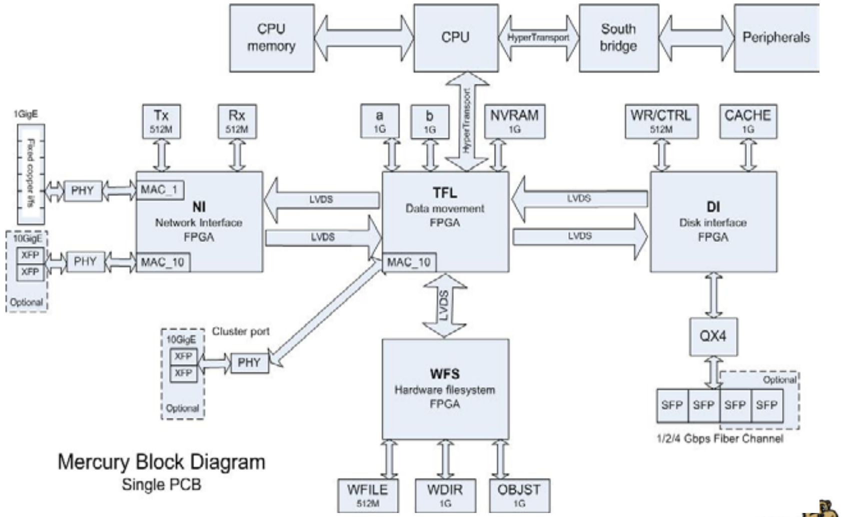
  Speedup = n/(LI+1)

- If one node's IO request takes 2 seconds (Tmax), while the remaining 999 node's IO request time (Tavg) averages 1.5, then the speedup over the 1000 processors is reduced to 750

Software-Hardware Co-Design of First Tier Storage Node

- New hardware technologies (SSD, hybrid DRAM) pushing the limits of OS storage/networking stack
- Question: Is it possible to co-design custom hardware and software for first-tier storage node?
- Example: File system in VLSI: Hitachi HNAS
    - design FPGA for specific





Mercury Block Diagram
Single PCB

Using FLASH in NAS Devices

- SpecSFS Benchmark #1: standard 15k FC disk setup, 224 drives, two NTAP 3160 filers.
  - Result was about 60k IOPS, ORT of 2.18 msec.
- SpecSFS Benchmark #2: basically the same setup, but replace those 15k drives with 7200rpm SATA + PAM cards. 96 drives in this case, and two PAM cards of 256GB each.
  - The result is nearly identical at 60k IOPS and ORT of 2.18 msec as well.
- Replaced lots of expensive 15k drives with fewer (and cheaper) 7200rpm drives if you just add a little bit of flash memory for metadata processing.

Nibbler: Co-Design Hardware and Software

- Accelerates memcached and first-tier storage node performance via SSD's and hybrid DRAM
  - Hardware-software codesign
  - e.g., BlueArc/HDS puts file system in VLSI

- Large, somewhat volatile memory
- Performance first, but also power and density
- First-tier storage node will drive ultimate performance achievable by this design

Questions?