HPC and Hierarchical Data Structures

IMAP Workshop
"Model and Data Hierarchies for Simulating and Understanding Climate"
May 2010

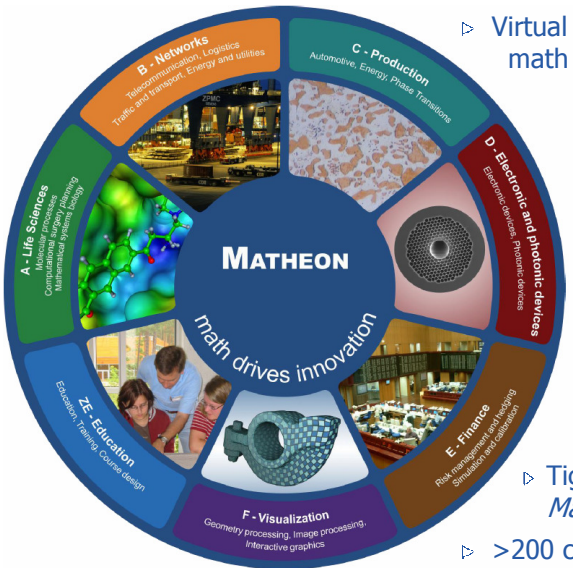
Tim Conrad
Freie Universität Berlin & MATHEON

DFG Research Center MATHEON
Mathematics for key technologies

Freie Universität Berlin

2010/05/14

MATHEON



- Virtual institute of the five leading math institutes in Berlin
  - Freie Universität Berlin
  - Humboldt-Universität zu Berlin
  - TU Berlin
  - WIAS
  - ZIB
- ~12.5 Mio EUR annual budget
- 60 projects in 7 areas
- 200 members
  - Mostly mathematicians
  - 41 Professors
  - 71 PhD Students
- Tightly coupled to *Berlin Mathematical* (Grad) *School*
- >200 cooperation partners
  - Industry, other institutes, …

Earth System Grid (ESG)

To support the infrastructural needs of the national and international climate community, ESG is providing crucial technology to securely access, monitor, catalog, transport, and distribute data in today's Grid computing environment.

Slide by Bernholdt

Why High-Performance Computing?

- Higher speed (solve problems faster).
  Important when there are "hard" or "soft" deadlines; e.g., 24-hour weather forecast
- Higher throughput (solve more problems)
  Important when there are many similar tasks to perform; e.g., transaction processing
- Higher computational power (solve larger problems) e.g., weather forecast for a week rather than 24 hours, or with a finer mesh for greater accuracy

Climate research and High Performance Computing (HPC)

- Part I:
  Introduction to HPC
- Part II:
  Illustrative Example
  Parallel Mesh Construction and Parallel Geometric Multigrids
- Part III:
  Data Storage: Getting data in and out

Parallelism means doing multiple things at the same time: you can get more work done in the same time.

Less fish ...

More fish!

Example modified from slides by Paul Gray

Introduction to Parallel Computing

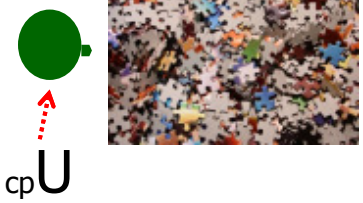
Part I

But there are some issues…

THE JIGSAW PUZZLE ANALOGY

Example modified from slides by Paul Gray

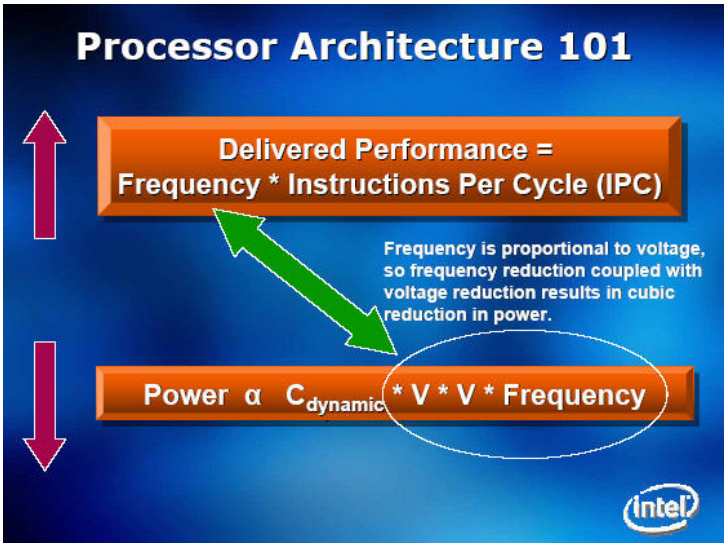
Serial Computing



- Suppose you want to do a jigsaw puzzle that has, say, a thousand pieces.
- We can imagine that it'll take you a certain amount of time. Let's say that you can put the puzzle together in an hour.

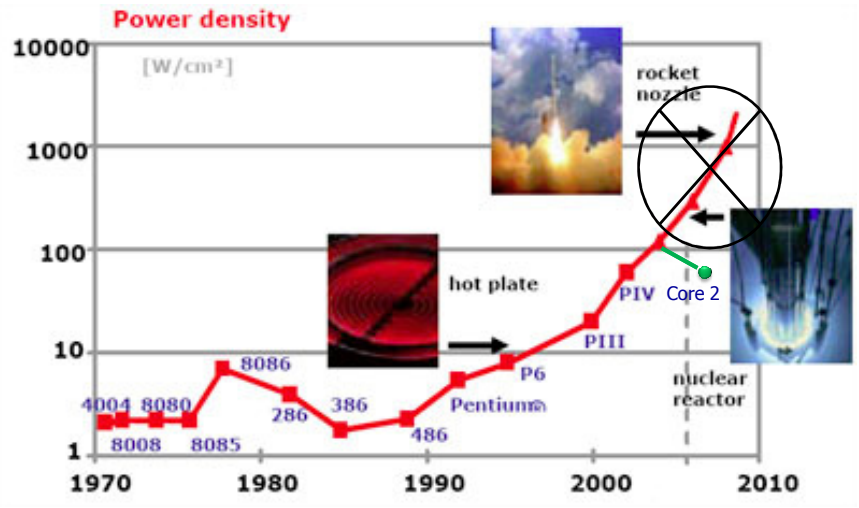
Can you do it in less time by thinking faster?

Why Not Crank Up the Clock?



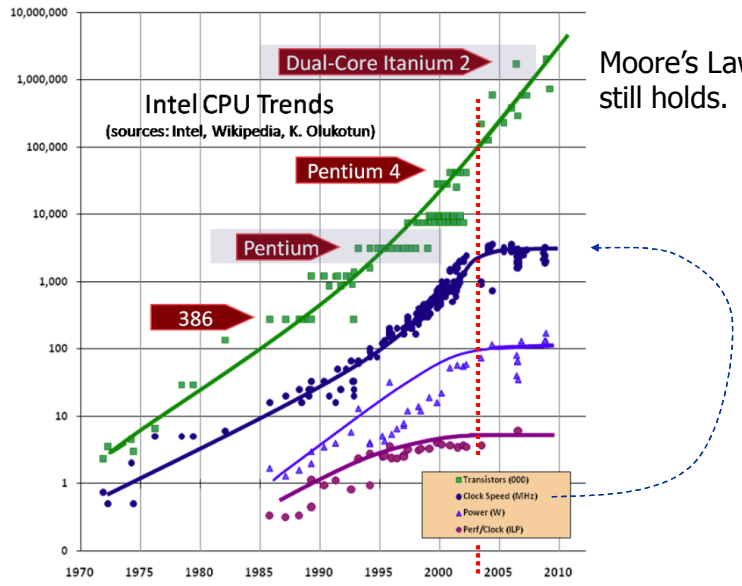
http://download.intel.com/technology/eep/fall_microprocessor_forum_2006.pdf

The Current Power Trend



Source: Intel Corp.

Obstacles, and Why You Don't Have 10GHz Today



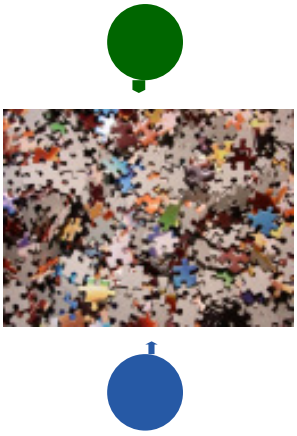
Moore's Law still holds.

Source: http://www.gotw.ca/publications/concurrency-ddj.htm

So… not a good idea. What's next?

SHARE THE WORKLOAD.

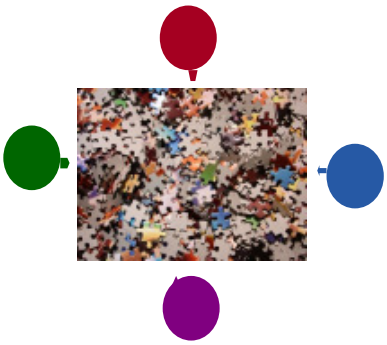
Shared Memory Parallelism



- If Julie sits across the table from you, then she can work on her half of the puzzle and you can work on yours.
- Once in a while, you'll both reach into the pile of pieces at the same time (you'll contend for the same resource), which will cause a little bit of slowdown.
- And from time to time you'll have to work together (communicate) at the interface between her half and yours. The speedup will be nearly 2-to-1: you all might take 35 minutes instead of 30.

Tim Conrad 13

The More the Merrier?



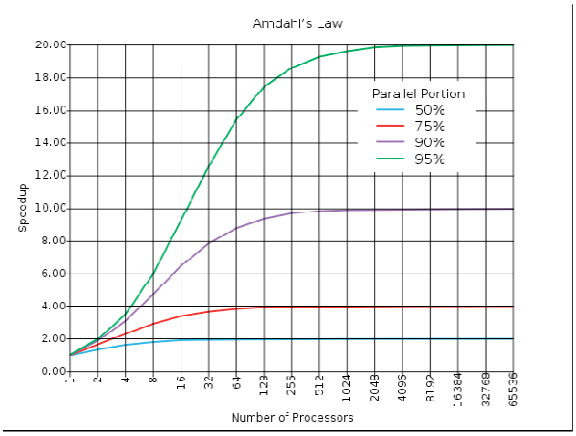
- Now let's put Lloyd and Jerry on the other two sides of the table.
- Each of you can work on a part of the puzzle, but there'll be a lot more contention for the shared resource (the pile of puzzle pieces) and a lot more communication at the interfaces.
- So you all will get noticeably less than a 4-to-1 speedup, but you'll still have an improvement, maybe something like 3-to-1:
- The four of you can get it done in 20 minutes instead of an hour.

Tim Conrad 14

Diminishing Returns

- If we now put Dave and Paul and Tom and Charlie on the corners of the table, there's going to be a whole lot of contention for the shared resource, and a lot of communication at the many interfaces.
- So the speedup you all get will be much less than we'd like; you'll be lucky to get 5-to-1.
- So we can see that adding more and more workers onto a shared resource is eventually going to have a diminishing return.

Challenge #1: Amdahl's Law



P = fraction of the work that can be parallelized

1-P = remainder, which cannot be parallelized

N: Number of CPU cores

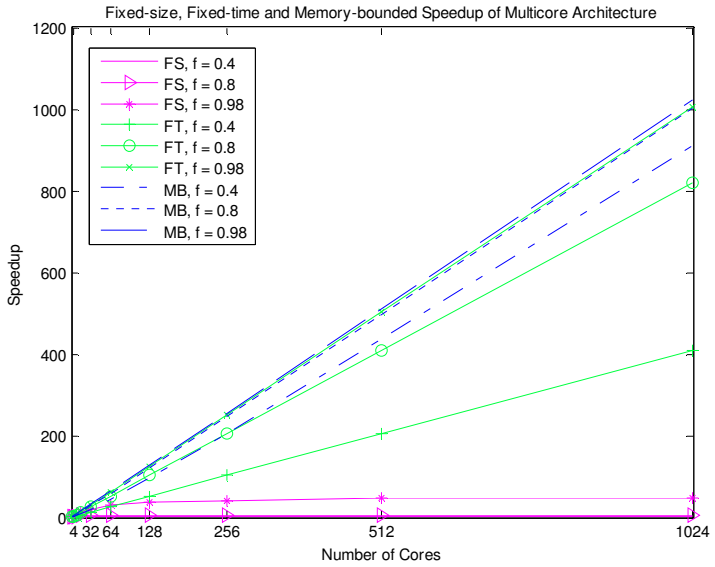
S = $\frac{1}{(1-P)+\frac{P}{N}}$.

The serial fraction of work limits the maximum speedup!

N.B.: Luckily, in many very large problems: P -> 1

Image: Wikipedia

Amdahl's Law Revisited: Speed-up for Multi-core

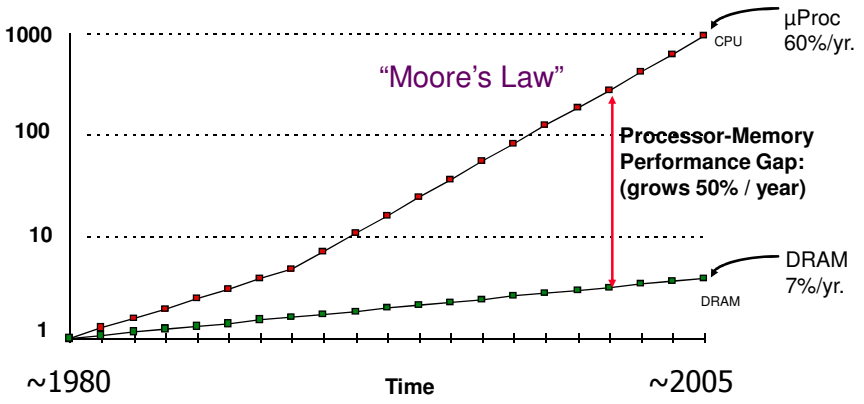


This is why super-computer exist!

Tim Conrad 17

Challenge #2: The Memory Wall

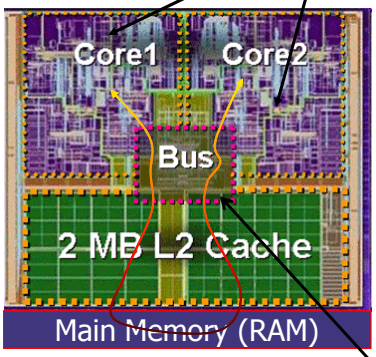
Communication in multi-core CPUs is done through memory.
But memory is not very fast – compared to the CPU.



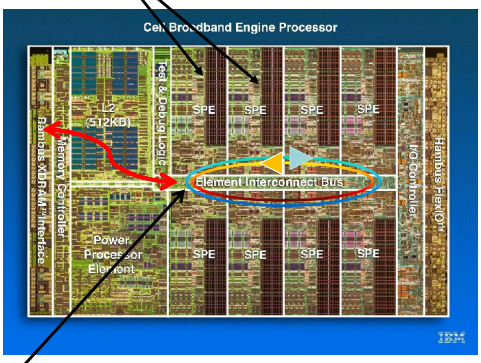
Tim Conrad 18

Communication by Shared-Memory

Programming Models and Compilers?
(-> handling shared resources)



Source: Intel Corp.



Source: IBM

Interconnection networks to
access main memory

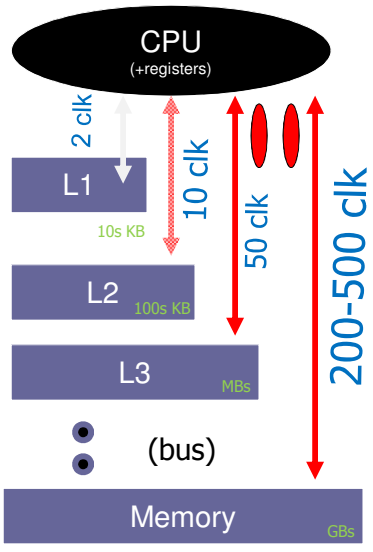
The Need for a Memory Hierarchy

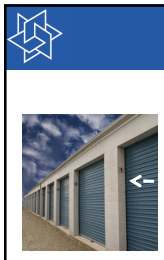
The widening speed gap between CPU and main memory

- Processor operations take of the order of 1 ns (~2clk @ 2GHz)
- Memory access requires 10s or even 100s of ns

Memory bandwidth/latency limits the instruction execution rate

- Each instruction executed involves at least one memory access
- Hence, a few to 100s of MIPS is the best that can be achieved
- A fast buffer memory can help bridge the CPU-memory gap
- The fastest memories are expensive and thus not very large.

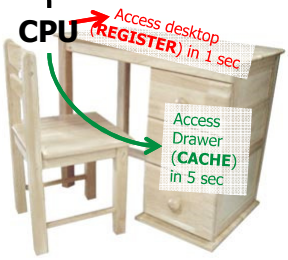




Office Analogy

Access storage building (HDD) in **weeks**

Once the "working set" (e.g. program loop) is in the drawer, very few trips to the file cabinet are needed.



different room ("off chip") very close ("on chip")

Example modified from B. Pahami's Book, "Computer Architecture"

Sharing is a good idea. Sharing a limited resource (memory) is not.

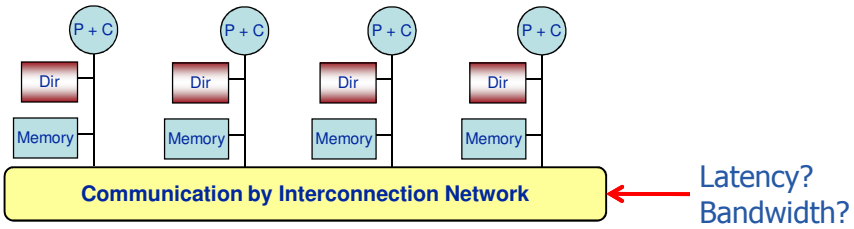
SHARE THE WORKLOAD ATTEMPT #2.

Distributed Parallelism

- Now let's try something a little different.
- Let's set up two tables, and let's put you at one of them and Julie at the other.
- Let's put half of the puzzle pieces on your table and the other half of the pieces on Julie's.
- Now you all can work completely independently, without any contention for a shared resource.
- **BUT**, the cost of communicating is **MUCH** higher (you have to scootch your tables together), and you need the ability to split up (decompose) the puzzle pieces reasonably evenly, which may be tricky to do for some puzzles.

Tim Conrad 23

MIMD Machines



- Parallel processing has catalyzed the development of a several generations of parallel processing machines
- Unique features include the interconnection network, support for system wide synchronization, and programming languages/compilers

Tim Conrad 24

Comparison of Switched Media

Type	Latency	Bandwidth	Cost
Gigabit Ethernet	~1 msec	0.1 GigaByte/sec	~ 50USD / port
10 Gigabit Ethernet	~100 μsec	1.0 GigaByte/sec	~ 500USD / port
QDR InfiniBand	~1μsec	3.6 GigaByte/sec	~ 1000USD / port

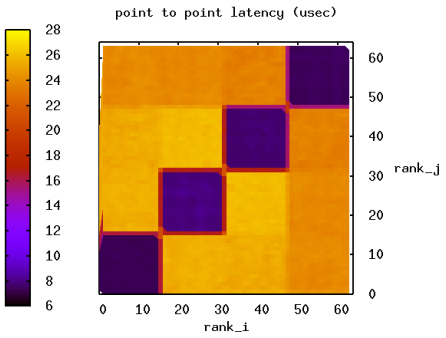
For comparison: RAM bandwidth is about 4GB/s (DDR4) - 28GB/s (XDR); latency ~0.02μs

Mellanox 36-port InfiniBand switch

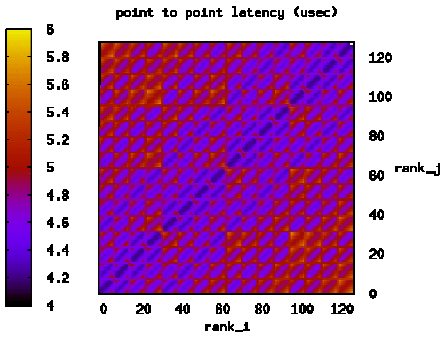
Notes about TCP/IP (window based):

- Protocol settings can greatly affect actual throughput! (e.g. only using some %)
- At 10 Gbps network speed, new packets arrive faster than current standard systems can process a packet. This increases the likelihood of dropped packets and defeats the value of providing greater bandwidth to a server
- For more, see tutorial at: http://psc.edu/networking/projects/tcptune/

Even today's machines are interconnect topology sensitive



Four IBM Power 3 nodes (16 processor) with Colony switch



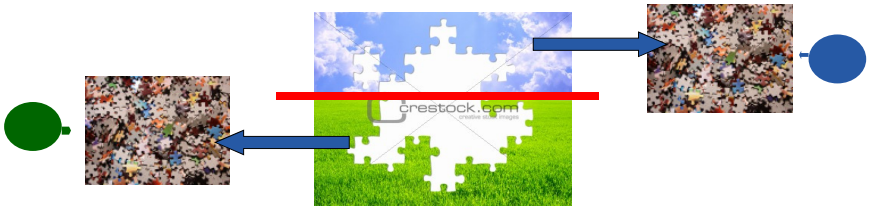
Interconnect Topology BG/L

More Distributed Processors

- It's a lot easier to add more processors in distributed parallelism.
- But, you always have to be aware of the need to decompose the problem and to communicate between the processors.
- Also, as you add more processors, it may be harder to load balance the amount of work that each processor gets.

Tim Conrad 27

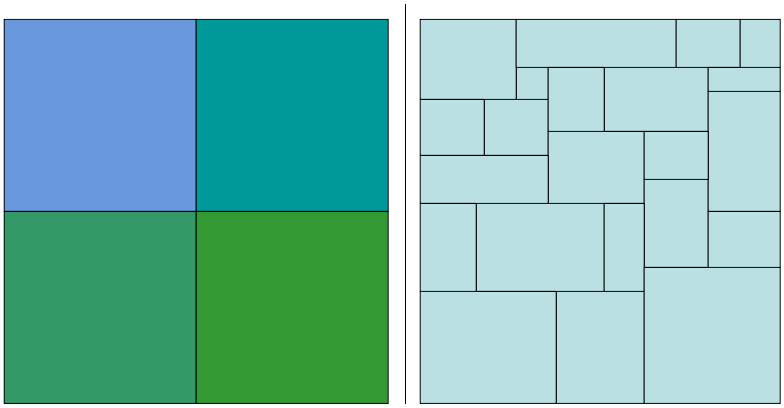
Load Balancing



- Load balancing means giving everyone roughly the same amount of work to do.
- For example, if the jigsaw puzzle is half grass and half sky, then you can do the grass and Julie can do the sky, and then you all only have to communicate at the horizon – and the amount of work that each of you does on your own is roughly equal.
- So you'll get pretty good speedup.

Tim Conrad 28

Load Balancing



Load balancing can be easy, if the problem splits up into chunks of roughly equal size, with one chunk per processor. Or load balancing can be very hard.

What do you load-balance?

Data-Parallel Approach

- Partition the data among the processors
- Each processor will execute the same set of commands

Control-Parallel Approach

- Partition the tasks to be performed among the processors
- Each processor will execute different commands

Hybrid Approach

- Switch between the two approaches at different stages of the algorithm
- Most parallel algorithms fall in this category

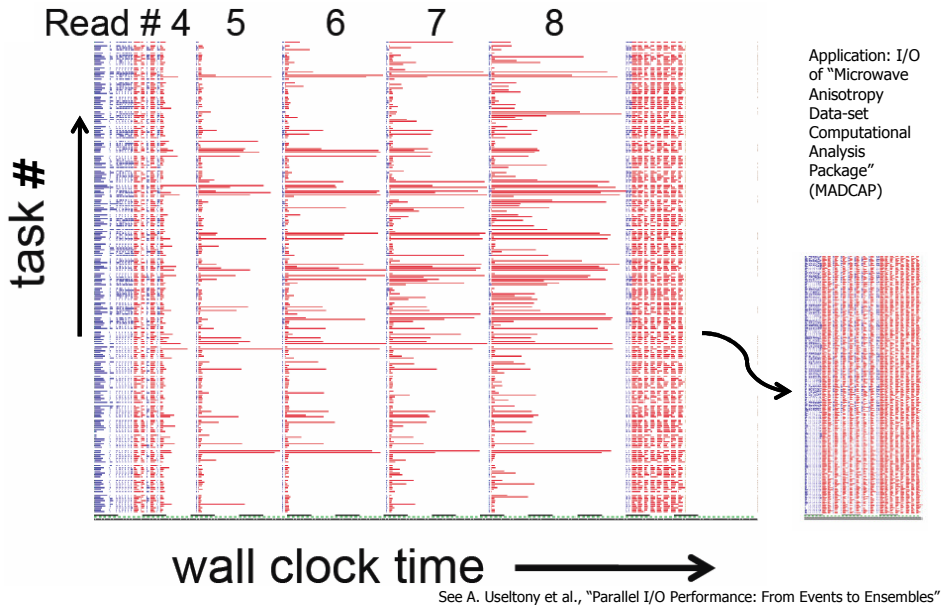
Example: Data-parallel Load Balancing

1. You request a certain number of processors
2. You setup a communicator
   - Give a unique id to each processor – rank
3. Every processor executes the same program
4. Communication is needed to scatter/gather data

Inside the program

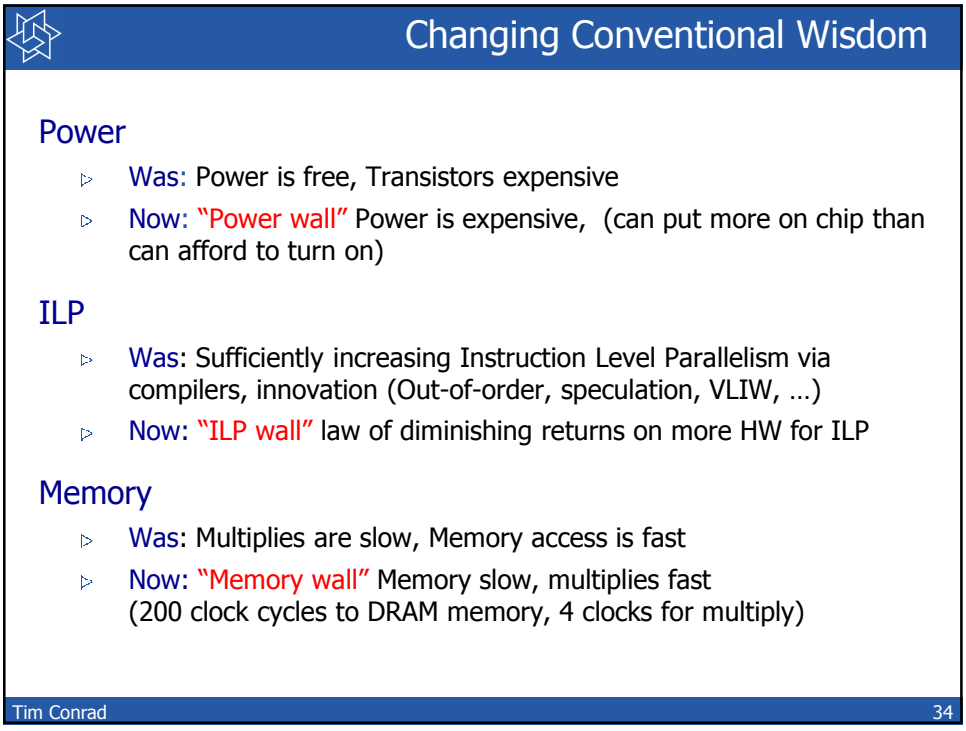
- Query for the rank and use it decide what to do
- Exchange messages between different processors using their ranks
- In theory, you only need 3 functions: Isend, Irecv, wait
- In practice, you can optimize communication depending on the underlying network topology – Message Passing Standards
  - E.g. Message Passing Interface (MPI, distributed computing) or
  - Open Message Passing (OpenMP – shared memory)

Load Balancing: IPM-I/O Trace



See A. Useltony et al., "Parallel I/O Performance: From Events to Ensembles"


End of Part I
LESSONS LEARNED
Tim Conrad
33


Changing Conventional Wisdom
Power
Was: Power is free, Transistors expensive
Now: "Power wall" Power is expensive, (can put more on chip than can afford to turn on)
ILP
Was: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
Now: "ILP wall" law of diminishing returns on more HW for ILP
Memory
Was: Multiplies are slow, Memory access is fast
Now: "Memory wall" Memory slow, multiplies fast
(200 clock cycles to DRAM memory, 4 clocks for multiply)
Tim Conrad
34

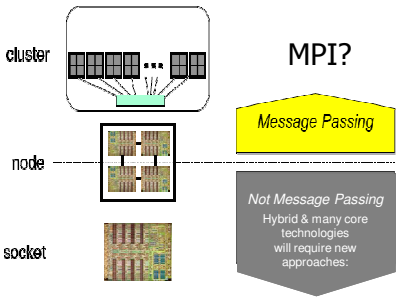
Changing Conventional Wisdom

Consequence

- Was: Uniprocessor performance doubles every 1.5 yrs
- Now:
  - Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now doubles about every 5(?) yrs
  - Sea change in chip design: multiple "cores" (#cores per chip will double every 2 years)
  - Simpler and slower processors are more power efficient
  - Large clusters of heterogeneous multi/many-core chips emerge

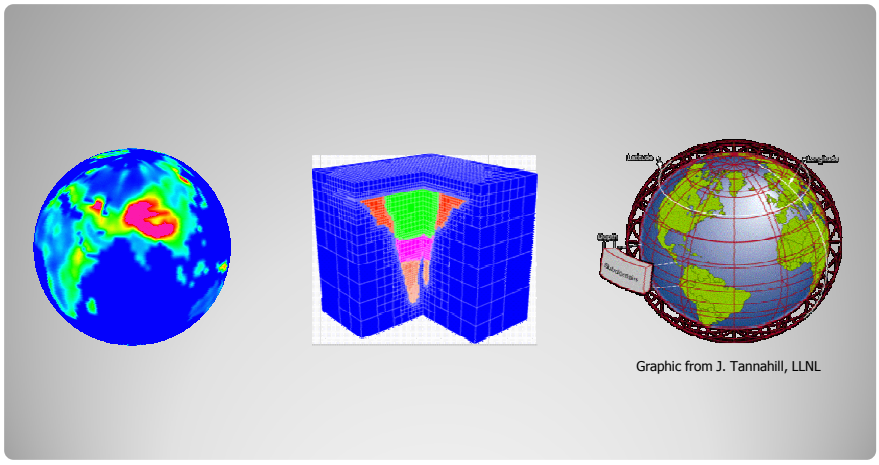
Example: Physics on GPU

- Need to deal with systems with millions of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism
- Different levels of parallelism (How to map algorithm-levels to hardware levels?)



after Don Grice, IBM, Roadrunner Presentation, ISC 2008

Tim Conrad



Graphic from J. Tannahill, LLNL

Mesh Construction and Parallel Geometric Multi Grids

Part II

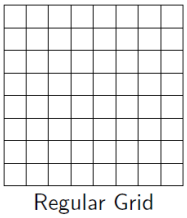
Tim Conrad 36

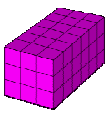
Motivation

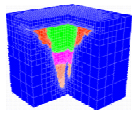
"Ironically, as numerical analysis is applied to larger and more complex problems, non-numerical issues play a larger role. Mesh generation is an excellent example of this phenomenon. Solving current problems in structural mechanics or fluid dynamics with finite difference of finite element methods *depends on the construction of high-quality meshes of surfaces and volumes*. *Geometric design and construction of these meshes are typically much more time-consuming than the simulations that are performed with them*."

- John Guckenheimer, "Numerical Computation in the Information Age" in June 1998 issue of SIAM News.

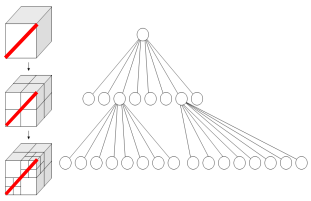
Adaptivity





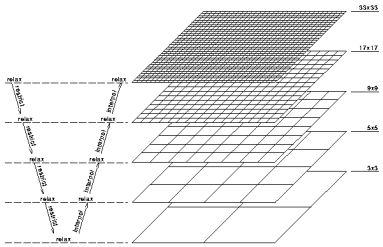


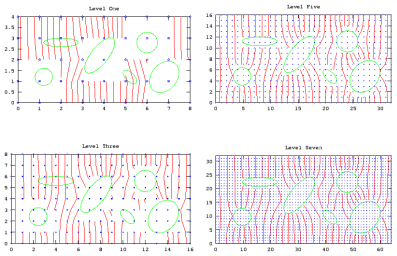
- Regular grids: Low overhead, more elements
- Unstructured grids: high overhead, fewer elements
- Octrees: good compromise in between



Images: Wikipedia, http://www.siw.com/examples_files/PitOct.gif, http://www.bugman123.com/Engineering/Engineering.html

Multi Grid Methods





Multi grid methods are optimal order solution algorithms for equation systems stemming from the discretization of (elliptic) PDEs (e.g. Laplace eq., Poisson eq.). They require linear time, i.e. *O(n)* operations for *n* unknowns.

Images from: http://www.mgnet.org/mgnet/tutorials/xwb/xwb.html

See e.g. "A Multi Grid Tutorial" by W. Briggs; https://computation.llnl.gov/casc/people/henson/mgtut/ps/mgtut.pdf
or http://www.fou.uib.no/fd/1996/h/413003/node47.html
or http://wissrech.ins.uni-bonn.de/research/projects/zumbusch/hash.html

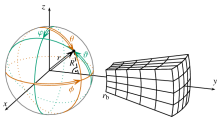
Structured Grids

NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2010; **17**:325–342
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla.699

Parallel geometric multigrid for global weather prediction

Sean Buckeridge*,† and Robert Scheichl

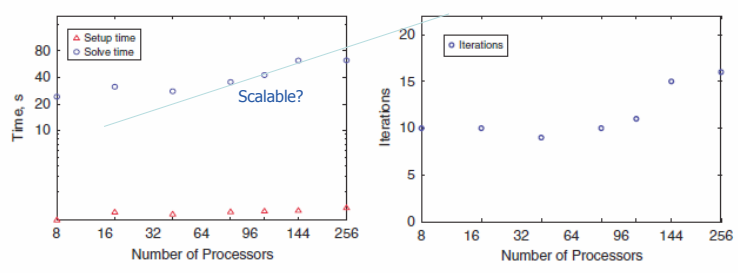
Department of Mathematical Sciences, University of Bath, Claverton Down, Bath BA2 7AY, U.K.



SUMMARY

The subject of this work is an optimal and scalable parallel geometric multigrid solver for elliptic problems on the sphere, crucial to the forecasting and the data assimilation tools used at the U.K. Met office. The optimality of multilevel techniques for elliptic problems makes them a suitable choice for these applications. The Met office uses spherical polar grids which, although structured, have the drawback of creating strong anisotropies near the poles. Moreover, a higher resolution in the radial direction introduces further anisotropies, and so modifications to the standard multigrid relaxation and the coarsening procedures are necessary to retain optimal efficiency. As the strength of anisotropy varies, we propose a non-uniform strategy, coarsening the grid only in regions that are sufficiently isotropic. This is combined with line relaxation in the radial direction. The success of non-uniform coarsening strategies has been demonstrated with algebraic multigrid (AMG) methods. Without the large setup costs required by AMG, however, we aim to surpass them with the geometric approach. We demonstrate the advantages of the method with experiments on model problems, both sequentially and in parallel, and show robustness and optimal efficiency of the method with constant convergence factors of less than 0.1. It substantially outperforms Krylov subspace methods with one-level preconditioners and the BoomerAMG implementation of AMG on typical grid resolutions. The parallel implementation scales almost optimally on up to 256 processors, so that a global solve of the quasi-geostrophic omega-equation with a maximum horizontal resolution of about 10 km and 3×10^9 unknowns takes about 60 s. Copyright © 2010 John Wiley & Sons, Ltd.

Structured Grids

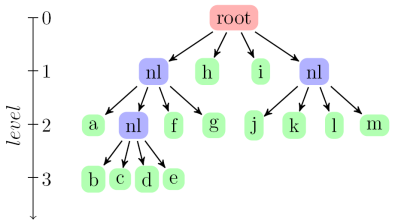


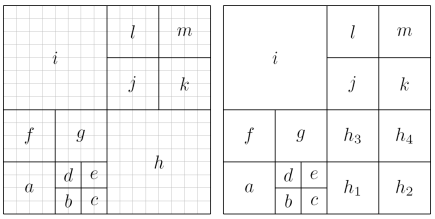
- Solving the Omega Equation
- Spherical polar grid
- Problem size: 192x120x50 (1.152.000) on each processor
- Stopping criterion: residual reduction of 10^{-8}

OCTREES

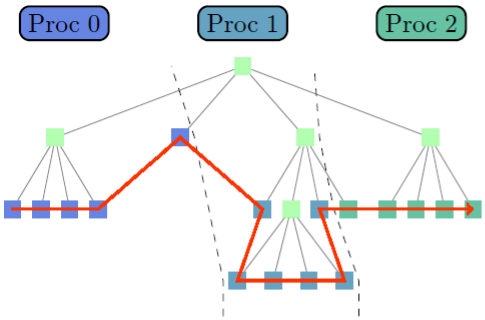
Linear Octree Data Structure

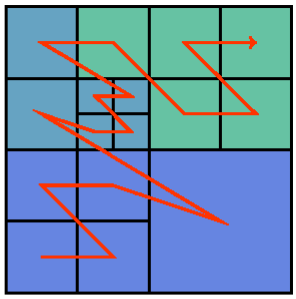
- Tree data structure used to store hierarchical information
  - Binary-trees – 1D, Quad-trees – 2D, Octrees – 3D
- It's sufficient to store the leaves: Linear Octrees
- Leaves can serve as elements of a finite element mesh
- Morton Ordering (pre-order traversal): A way to sort leaves





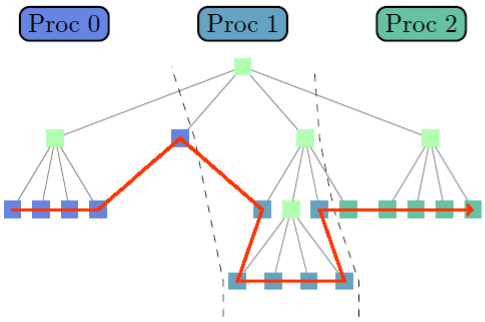
Octrees and Space filling curves (SFC)

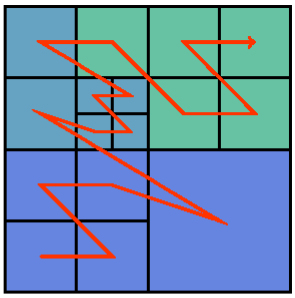




- 1:1 relation between octree and SFC → efficient encoding
- Map a 1D curve into 2D or 3D space → total ordering
- Recursive self-similar structure → scale-free
- Tree leaf traversal → cache-friendly

Octrees and Space filling curves (SFC)

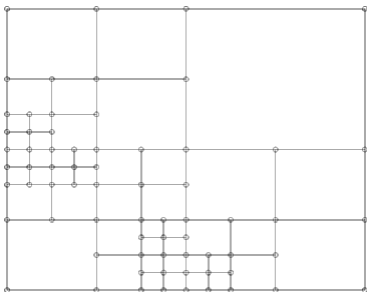


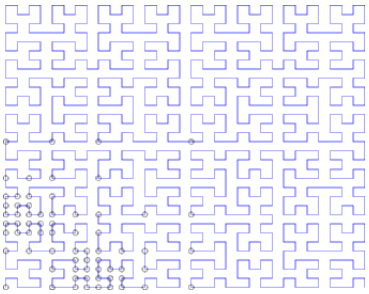


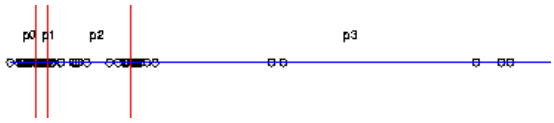
local information

- Find parent or children → vertical tree step $\mathcal{O}(1)$
- Find on-processor neighbor → tree search $\mathcal{O}(\log \frac{n}{p})$
- Find owner of off-processor neighbor → binary search $\mathcal{O}(\log p)$

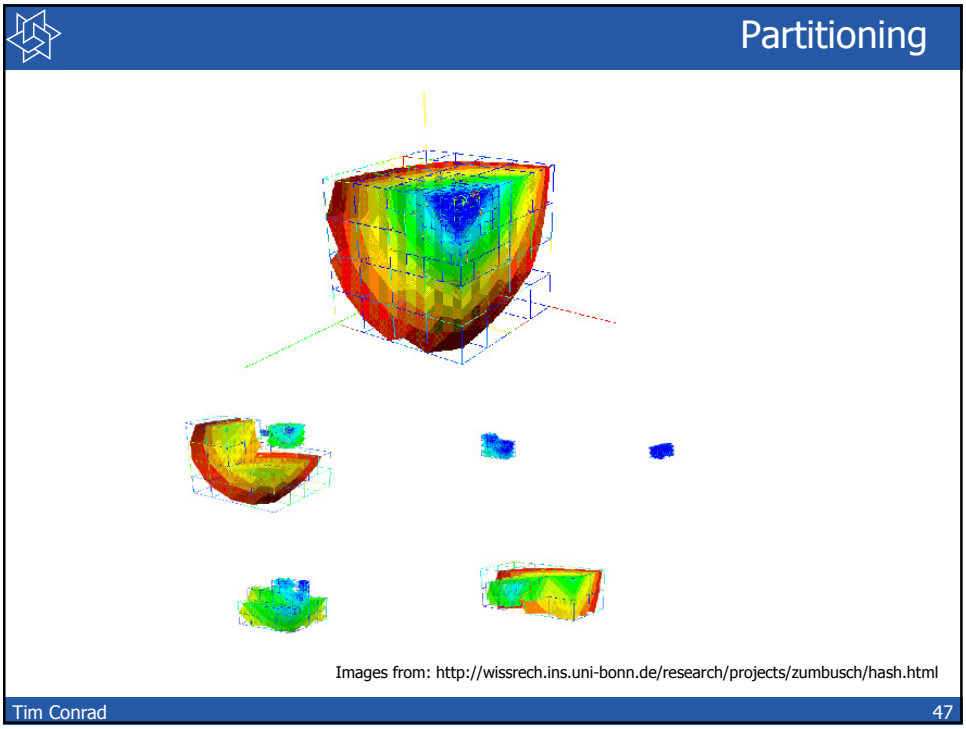
Easy Partitioning





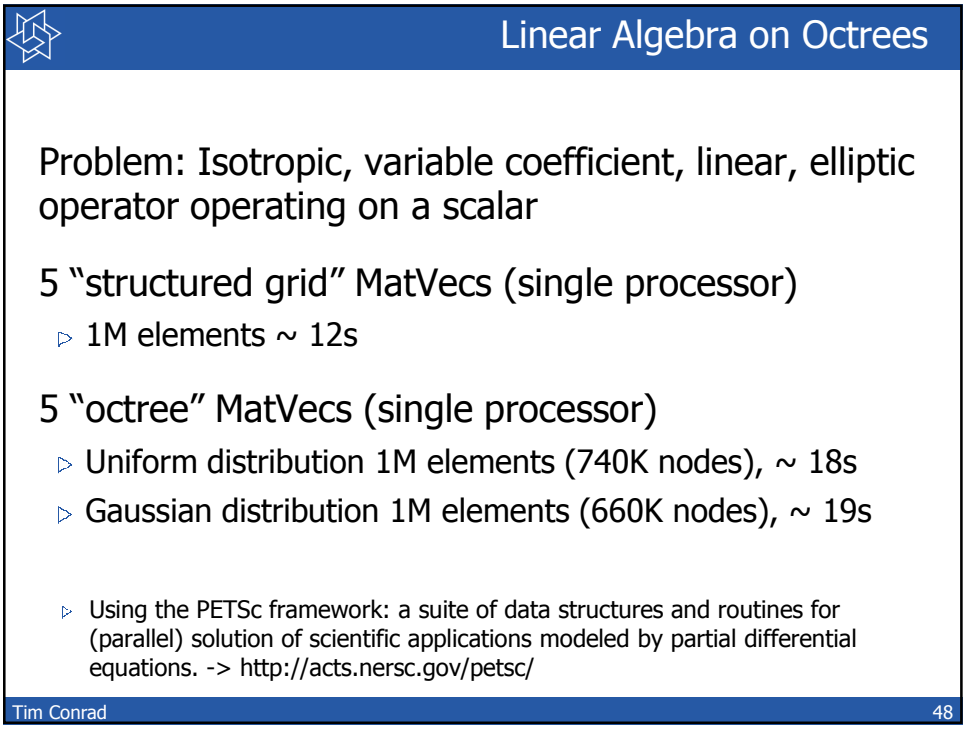


Images from: http://wissrech.ins.uni-bonn.de/research/projects/zumbusch/hash.html



Partitioning

Images from: http://wissrech.ins.uni-bonn.de/research/projects/zumbusch/hash.html



Linear Algebra on Octrees

Problem: Isotropic, variable coefficient, linear, elliptic operator operating on a scalar

5 "structured grid" MatVecs (single processor)

- 1M elements ~ 12s

5 "octree" MatVecs (single processor)

- Uniform distribution 1M elements (740K nodes), ~ 18s
- Gaussian distribution 1M elements (660K nodes), ~ 19s

- Using the PETSc framework: a suite of data structures and routines for (parallel) solution of scientific applications modeled by partial differential equations. -> http://acts.nersc.gov/petsc/

Dendro

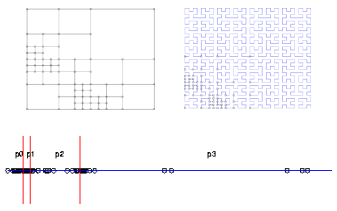
- A suite of parallel algorithms for the discretization and solution of partial differential equations that require discretization of second-order elliptic operators.
- It supports trilinear finite element discretizations constructed using octrees.
- Dendro has modules for
  - bottom-up octree generation and 2:1 balancing,
  - meshing,
  - geometric multi grids.
- It supports the PETSc objects 'Mat' and 'Vec' and provides interfaces to PETSc's linear and non-linear solvers.
- PETSc is a suite of data structures and routines for (parallel) solution of scientific applications modeled by partial differential equations.

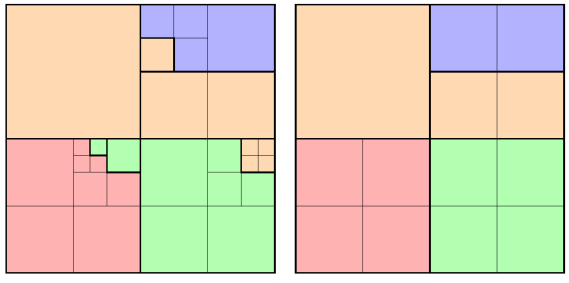
See also:
- R. Sampath, "Bottom-up construction and 2:1 Balance refinement of linear octrees in parallel", Univ. of Pennsylvania, Tech. Report, MS-CIS-07-05, 2007
- R. Sampath, "Low-constant parallel algorithms for finite element simulations using linear octrees", Supercomputing, November 2007
- M. Griebel, G. Zumbusch, Hash-Storage Techniques for Adaptive Multilevel Solvers and their Domain Decomposition Parallelization, Proceedings of Domain Decomposition Methods 10, 1998
- M. Griebel, G. Zumbusch, Parallel multigrid in an adaptive PDE solver based on hashing, Proceedings of ParCo '97, 1998

Outline of approach

Key ideas:

- Minimizing communication during meshing by 'Block Partition'
  - Performance gain: ~4fold compared to similar approaches
- New strategies for matvec operations by single tree traversal
- Minimizing storage overhead by mesh compression
  - 3fold by entropy encoding (Golomb-Rice)





Morton Partition **Block Partition**

The Dendro Framework

MULTIGRIDS WITH OCTREES

Dendro Algorithm

1. Given input points, construct and partition octree at the fine level.
2. Given the total number of levels, construct and partition the coarser octrees.
3. Mesh the octrees.
4. Construct restriction and prolongation operators.

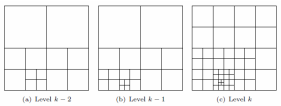


Figure 1: *Quadtree meshes for three successive multigrid levels.*

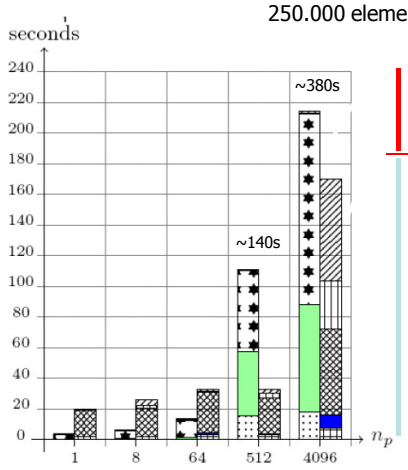
Outline of the Dendro algorithms. The main algorithm in Dendro is the geometric multigrid solver. In Dendro, we use a hierarchy of octrees (see Figure 1), which we construct as part of the geometric multigrid V-cycle algorithm. The V-cycle algorithm consists of 6 main steps: (1) Pre-smoothing: $u_k = S_k(u_k, f_k, A_k)$; (2) Residual computation: $r_k = f_k - A_k u_k$; (3) Restriction: $r_{k-1} = R_k r_k$; (4) Recursion: e_{k-1}=Multigrid(A_{k-1}, r_{k-1}); (5) Prolongation: $e_k = P_k e_{k-1}$; and (6) Post-smoothing: $u_k = S_k(u_k, f_k, A_k)$. When "k" reaches a minimum level, which we term the "*exact solve*" level, we solve for e_k exactly using a single level solver (e.g., a parallel sparse direct factorization method).

Octree Forest construction	Bottom-up Coarsening	Top-Down Meshing
1. Bottom-up construct fine octree	1. Replace leafs with their parent	1. Mesh and partition coarse grid
2. Partition fine octree	2. Balance and Morton-order partition	2. Prolong partitioning to next grid
3. Balance fine octree	3. Repeat	3. Check load balancing and repartition
4. Bottom-up coarsen octree		4. Construct prolongation operator
5. Top-down meshing		

Table 1: *Summary of main algorithmic components for the multigrid construction used in* Dendro *[19].*

Dendro Benchmarks

250.000 elements per CPU at finest grid



		1	8	64	512	4096
Coarsening		0.011	0.14	0.16	15.24	18.2
Balancing		0.32	0.72	1.34	42.39	69.8
Meshing		2.8	4.69	10.92	52.55	124.62
R-setup		0.38	0.38	0.55	0.55	0.61
Extras-1		0.1	0.16	0.48	0	1.16
LU		0.35	2.18	0.56	3.02	31.2
Restriction/ Prolongation		1.7	2.14	3.47	3.11	7.34
Scatter		0	0.048	0.68	0.69	9.02
FE Matvecs		17.32	18.2	26.63	23.68	56.04
Extras-2		0.513	3.35	1.19	2.29	66.7
Elements		220K	1.9M	15.3M	122.6M	980.6M
Vertices		152K	1.5M	13.7M	115.9M	953.8M
Meshes		5	9	13	15	15

Figure 3: *The LEFT FIGURE shows isogranular scalability with a grain size of 0.25M (approx) elements per CPU (n_p) on the finest level. The difference between the minimum and maximum levels of the octants on the finest grid is 5. A V-cycle using 4 pre-smoothing steps and 4 post-smoothing steps per level was used as a preconditioner to CG. The damped Jacobi method with a damping factor of 0.857 was used as the smoother at all multigrid levels. A relative tolerance of 10^{-10} in the 2-norm of the residual was used. 6 CG iterations were required in each case, to solve the problem to the specified tolerance. The finest level octrees for the multiple CPU cases were generated using regular refinements from the finest octree for the single CPU case. SuperLU was used to solve the coarsest grid problem. This isogranular scalability experiment was performed on NCSA's Intel 64 cluster.* (Tolerance: 10^{-10})

An essential element of computational science is IT infrastructure for managing the coming "data tsunami".

(E.g. LHC, genomics, climate simulations,…)

Transferring and Storing Data

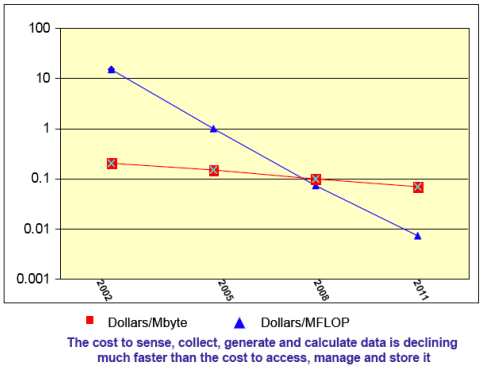
Part III

Challenge Data Intensive Computing

Our ability to sense, collect, generate and calculate on data is growing faster than our ability to access, manage and even "store" that data

Influences

- Sensing, acquisition, streaming applications
- Huge active data models
  - Biological modeling (Blue Brain)
  - Massive on line games
- Huge data sets
  - Medical applications
  - Astronomical applications
  - Climate applications
- Archiving
  - Preservation
  - Access
  - Legal requirements
- Systems technology
  - Computing in memory



Source: David Turek, IBM

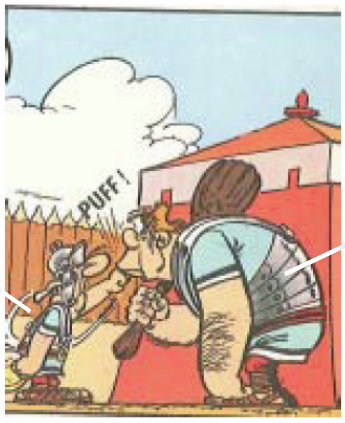
I/O in Computational Science

HOW DOES THAT AFFECT ME?

I/O in Computational Science

I/O – Input (e.g. initialization) usually not critical since (relatively) small

Your application transferring a small file.



Your (parallel) file system.

Example modified from W. Frings / G. Sutmann

... but ...

I/O in Computational Science

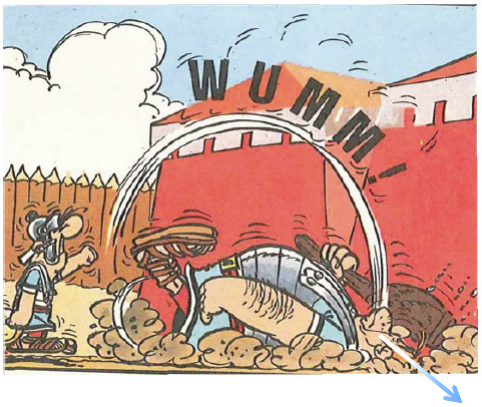
Scaling on large platforms might be become a problem (if serialized)

Many instances of your applications transferring small files...

... and ...

I/O in Computational Science

The effect can be dramatic and performance is degraded.



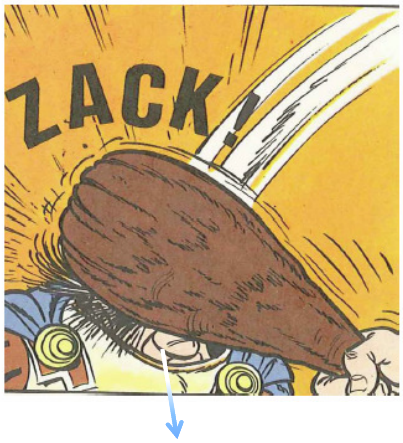
Your (parallel) file system.

I/O in Computational Science

Output is even MUCH worse since a lot larger.

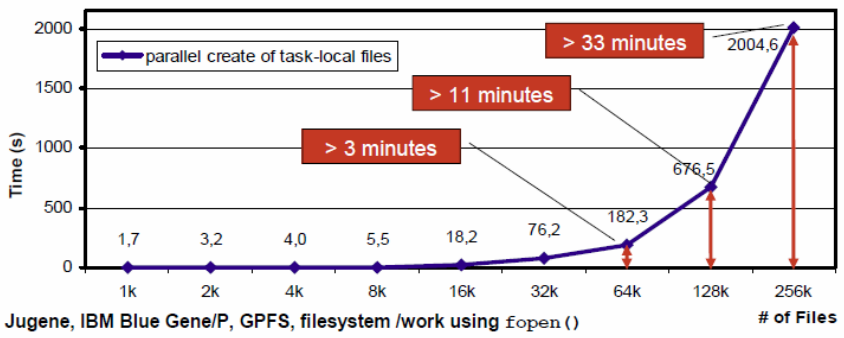
- E.g. Checkpointing
  - Millions of states in a trajectory: 10s of MBs
  - Thousands of trajectories: 100s of GBs
  - Lots of data written all at once, e.g. BlueGene with 512 compute nodes each having 60MB/s bandwidth:

    512MB/s * 60MB/s = 30GB/s



Again: your (parallel) file system.

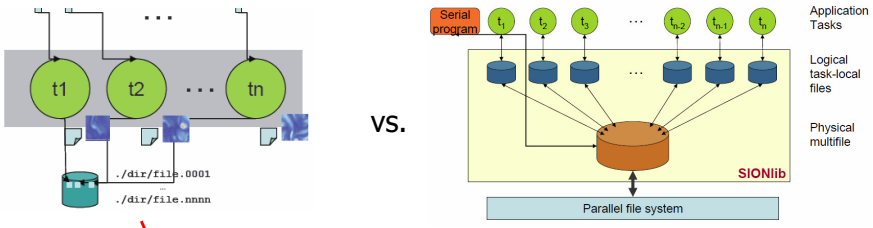
Creating files in parallel in the same directory



Problem: Contention at the FS meta data server. (Amdahl…)
(Similar results for other parallel FSs.)

W. Frings et al., "SIONlib: Scalable parallel I/O for task-local files"

Parallel I/O



I/O type	#tasks	trace size	activation	write BW
Task-local	65536	1105 GB	253.5 s	3476 MB/s
SIONlib	65536	1105 GB	25.9 s	3589 MB/s

10x faster

Contention

bandwidth	write	read
SIONlib (aligned)	5.4 GB/s	5.1 GB/s
MPI I/O (not aligned)	2.8 GB/s	3.0 GB/s

JUGENE: 16k tasks, 16 files, 16.5 MB/task

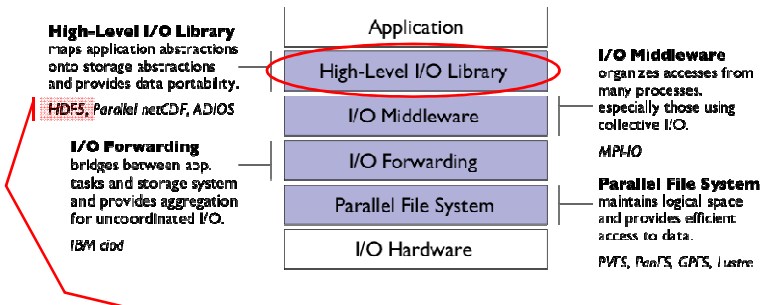
HDF5 / pNetCDF

W. Frings et al., "SIONlib: Scalable parallel I/O for task-local files"

I/O in Computational Science

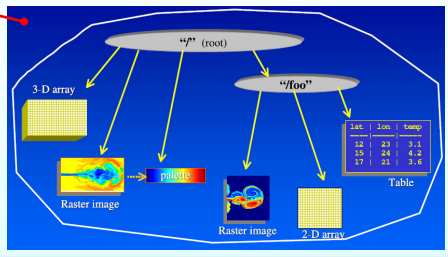
STORING USER DATA

The I/O Software Stack



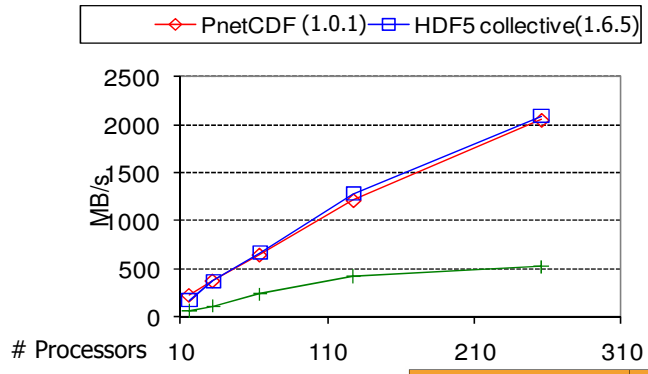
Structured data storage

- Multidimensional, hierarchical, typed datasets
- Metadata is placed in the file itself
- Simplifying data movement, archiving, …



HDF5 and PnetCDF performance comparison

Flash I/O Benchmark (Checkpoint files)



Flash: Parallel, adaptive-mesh simulation code (mainly astro-physics)

Benchmark by C. Chilan et al.: "Parallel I/O Performance Study and Optimizations with HDF5, A Scientific Data Package"

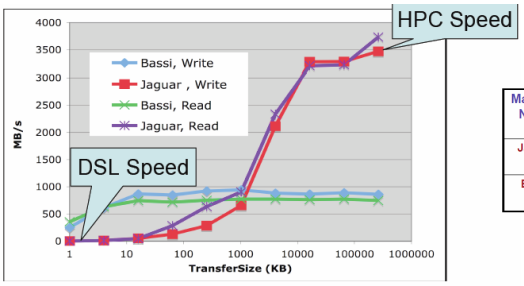
Parallel Environment Requirements

- MPI with MPI-IO
  - MPICH ROMIO
  - Vendor's MPI-IO
- Parallel file system
  - GPFS
  - PVFS
  - Lustre
  - Specially configured NFS

Parallel netCDF	Parallel HDF5
• Contiguous or interleaved data layout; • Sits on top of MPI-IO, transferring user access patterns directly to MPI fileview, little overhead. • One time header I/O gets all necessary info for direct access of each data array; • By maintaining a local copy of header, each process can access any array identified by its permanent ID at any time, without any collective open/close operation of the object. Saves a lot of inter-process communication.	• Tree-like file structure; • Use dataspace and hyperslabs to define data organization, map and transfer data between memory and filespace, pack or unpack. • Iterate through entire namespace to get the header info to access each object; • Need collective open/close operation when accessing each single object, and for file write, the metadata need to be updated in a synchronous way. So lots of communication overhead.

Need for Large Chunks

Parallel file systems and APIs on top of them are not DBs – they need large transfers, at least 64kB.



Machine Name	Parallel File System	Proc Arch	Inter-connect	Peak IO BW	Max Node BW to IO
Jaguar	Lustre	Opteron	SeaStar	18*2.3GB/s = 42GB	3.2GB/s (1.2GB/s)
Bassi	GPFS	Power5	Federation	6*1GB/s = ~6.0GB/s	4.0GB/s (1.6GB/s)

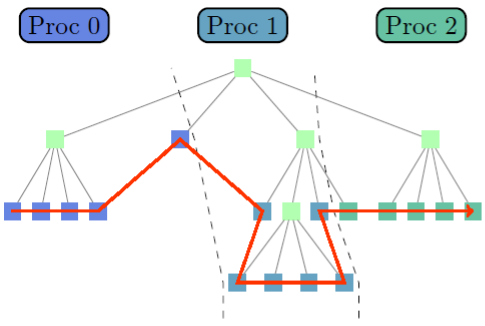
Image by H. Shan; https://secure.nersc.gov/projects/presentations/Shan_CUG07.pdf

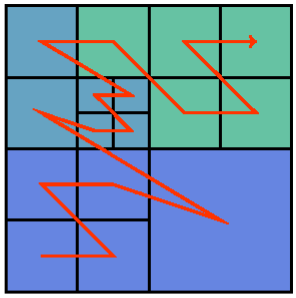
What if I don't have a parallel file system?
How to take the load off of the file system?

What if I DO have many small I/O operations?

DISTRIBUTED HASH-TABLES

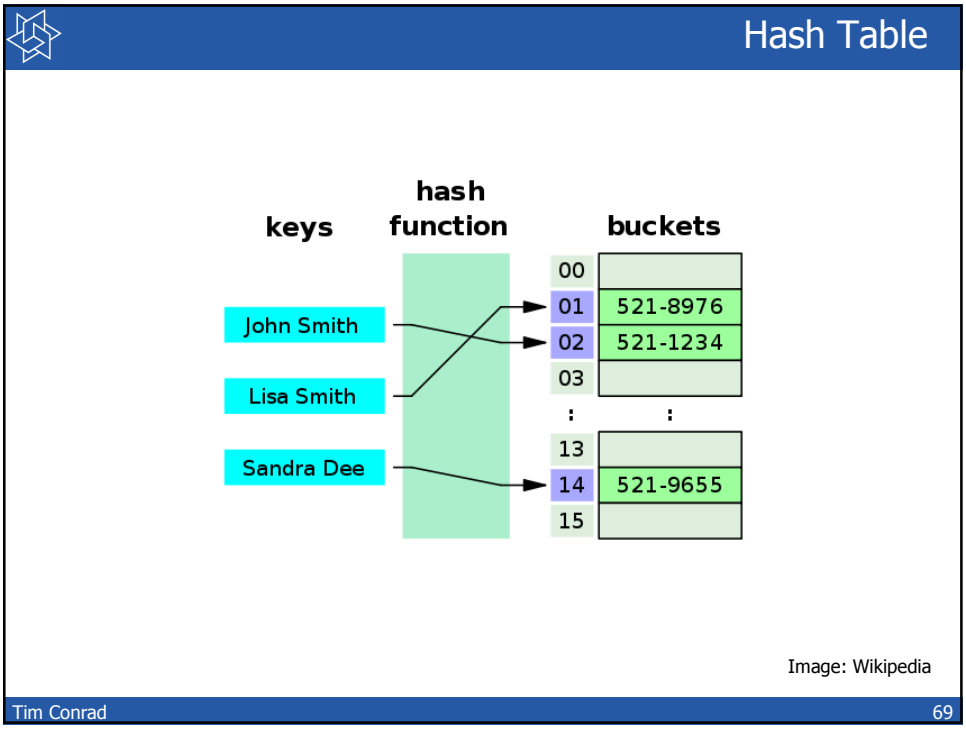
Octrees and Space filling curves (SFC)

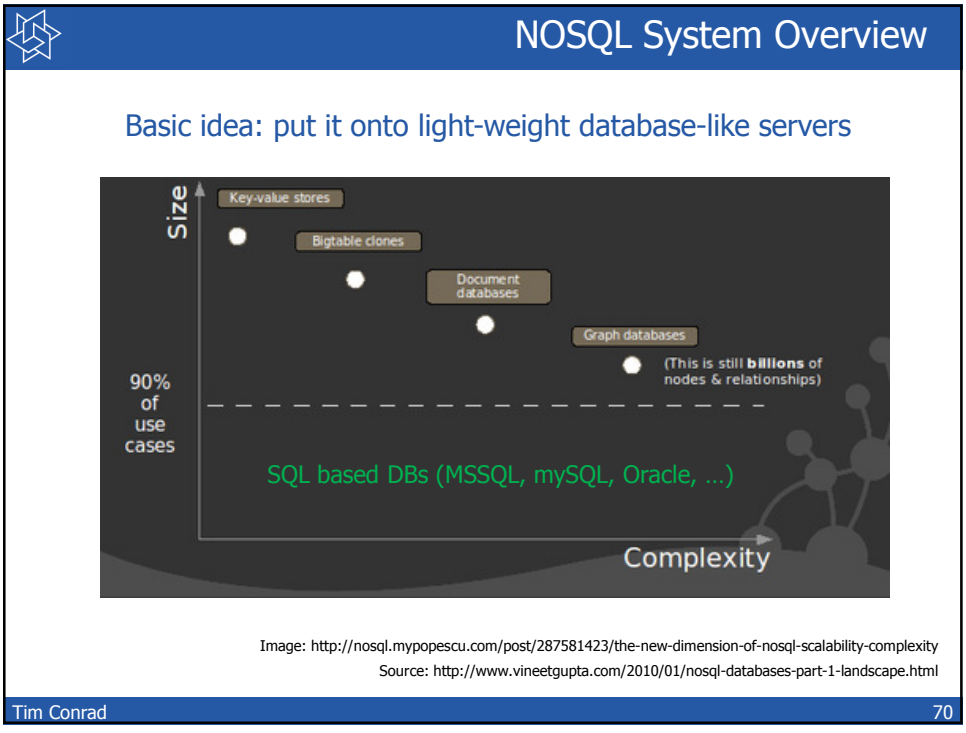




- 1:1 relation between octree and SFC → efficient encoding
- Map a 1D curve into 2D or 3D space → total ordering
- Recursive self-similar structure → scale-free
- Tree leaf traversal → cache-friendly

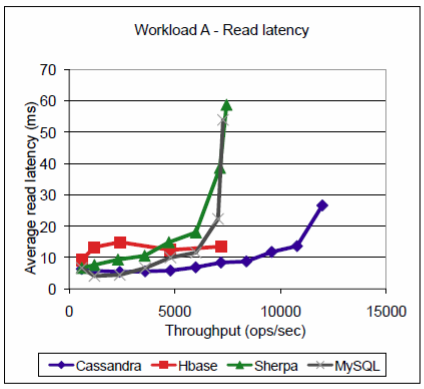
Each data point has unique "address"

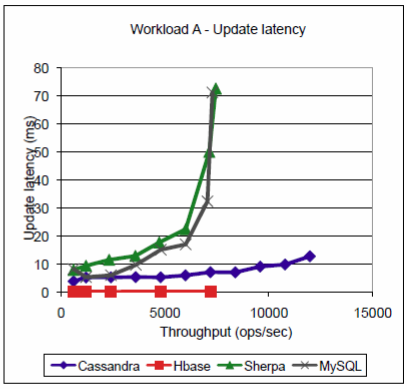

Hash Table
keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
521-8976
02
521-1234
03
13
14
521-9655
15
Image: Wikipedia
Tim Conrad
69


NOSQL System Overview
Basic idea: put it onto light-weight database-like servers
Size
Key-value stores
Bigtable clones
Document databases
Graph databases
(This is still billions of nodes & relationships)
90% of use cases
SQL based DBs (MSSQL, mySQL, Oracle, ...)
Complexity
Image: http://nosql.mypopescu.com/post/287581423/the-new-dimension-of-nosql-scalability-complexity
Source: http://www.vineetgupta.com/2010/01/nosql-databases-part-1-landscape.html
Tim Conrad
70

SQL-based Systems are too "Heavy"

- 50/50 Read/update

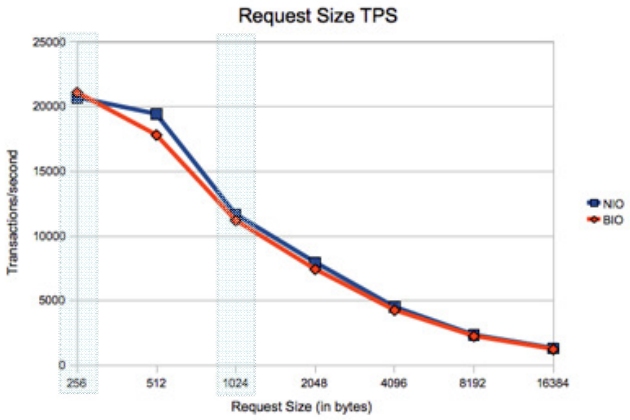




Comment: Cassandra is optimized for writes, and achieves higher throughput and lower latency. Sherpa and MySQL achieve roughly comparable performance, as both are limited by MySQL's capabilities. HBase has good write latency, because of commits to memory, and somewhat higher read latency, because of the need to reconstruct records.9

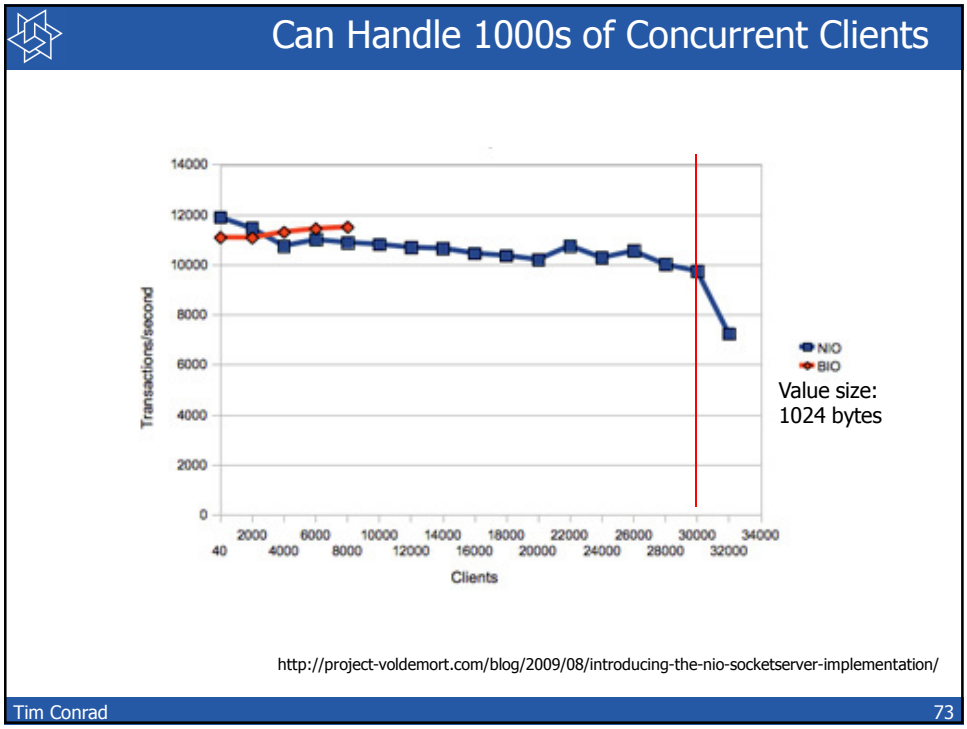
http://www.brianfrankcooper.net/pubs/ycsb-v4.pdf

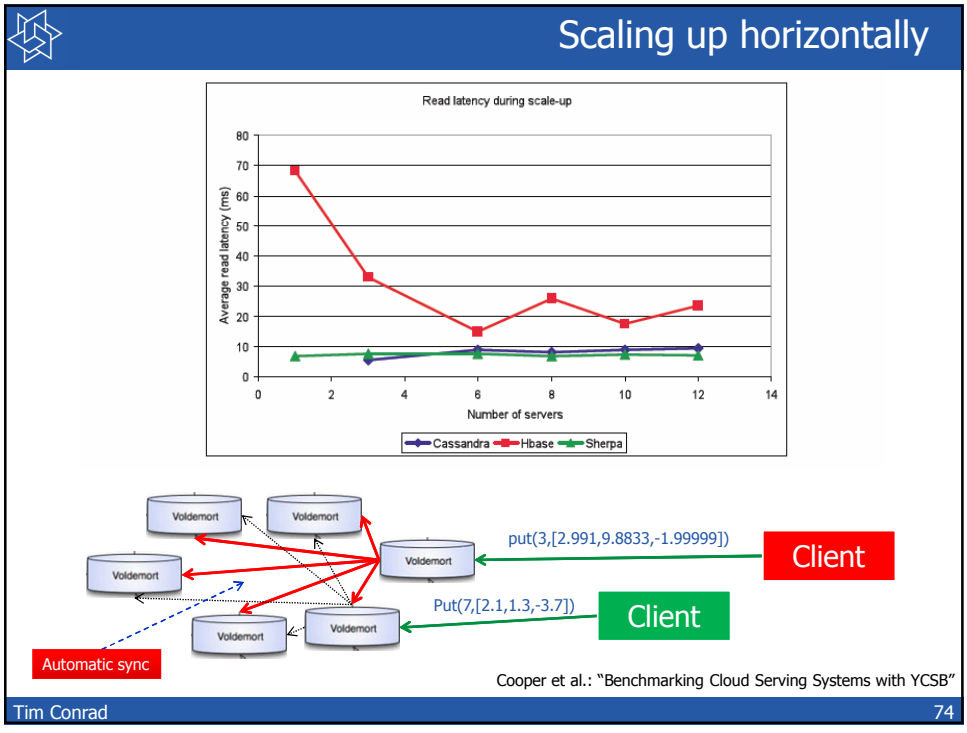
NOSQL Can Handle 1000s of Requests a Second

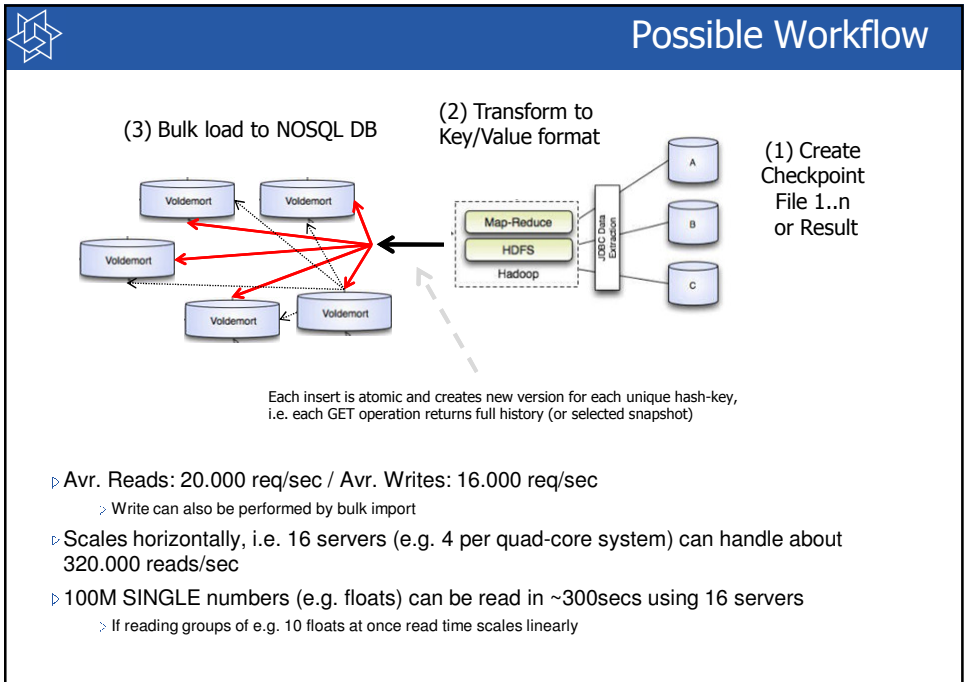


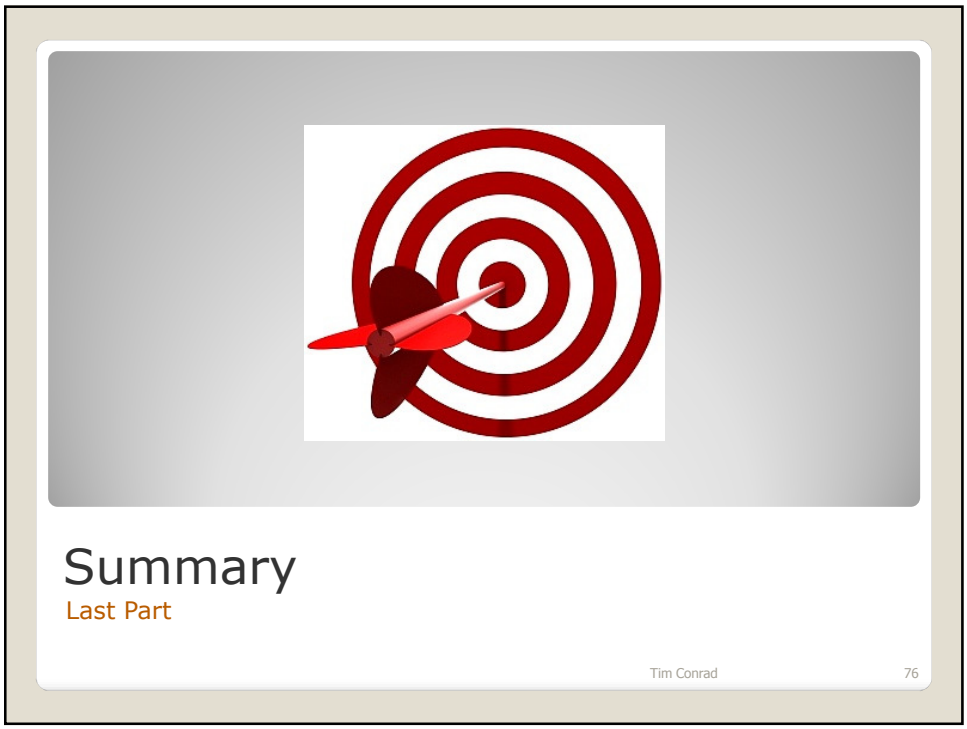
- Example: put(KEY,VALUE) - e.g.: put(3,[2.991,9.8833,-1.99999])
- Key can be e.g. position on space filling curve

http://project-voldemort.com/blog/2009/08/introducing-the-nio-socketserver-implementation/


Can Handle 1000s of Concurrent Clients
Transactions/second
Clients
NIO
BIO
Value size:
1024 bytes
http://project-voldemort.com/blog/2009/08/introducing-the-nio-socketserver-implementation/
Tim Conrad
73


Scaling up horizontally
Read latency during scale-up
Average read latency (ms)
Number of servers
Cassandra
Hbase
Sherpa
Voldemort
put(3,[2.991,9.8833,-1.99999])
Client
Put(7,[2.1,1.3,-3.7])
Client
Automatic sync
Cooper et al.: "Benchmarking Cloud Serving Systems with YCSB"
Tim Conrad
74


Possible Workflow
(3) Bulk load to NOSQL DB
(2) Transform to Key/Value format
(1) Create Checkpoint File 1..n or Result
Voldemort
Voldemort
Voldemort
Voldemort
Voldemort
Map-Reduce
HDFS
Hadoop
JDBC Data Extraction
A
B
C
Each insert is atomic and creates new version for each unique hash-key, i.e. each GET operation returns full history (or selected snapshot)
Avr. Reads: 20.000 req/sec / Avr. Writes: 16.000 req/sec
Write can also be performed by bulk import
Scales horizontally, i.e. 16 servers (e.g. 4 per quad-core system) can handle about 320.000 reads/sec
100M SINGLE numbers (e.g. floats) can be read in ~300secs using 16 servers
If reading groups of e.g. 10 floats at once read time scales linearly
Tim Conrad
75


Summary
Last Part
Tim Conrad
76

Summary

- Part I: Introduction to HPC
  - We are hitting a brick wall (= clock/memory/ILP wall) – new concepts for algorithmic design and their implementation are needed
  - Communication is expensive
- Part II: Illustrative Example
  - Octrees can be an alternative data structure for meshing and multi grid methods (if done right)
- Part III: Data Storage
  - We are producing more data than we can store
  - Parallel file systems are not the only answer
  - Need hierarchies / load-balancing even on file system level
  - Light-weight DB approaches can be an option

Thank you for your attention.

Tim Conrad

conrad@math.fu-berlin.de

Institut of Mathematics, Freie University Berlin & MATHEON

Open Source Parallel Software

PETSc (Linear and NonLinear Solvers)
- http://www-unix.mcs.anl.gov/petsc/petsc-as/

ScaLAPACK (Linear Algebra)
- http://www.netlib.org/scalapack/scalapack_home.html

SPRNG (Random Number Generator)
- http://sprng.cs.fsu.edu/

Paraview (Visualization)
- http://www.paraview.org/HTML/Index.html

NAMD (Molecular Dynamics)
- http://www.ks.uiuc.edu/Research/namd/

CHARMM++ (Parallel Objects)
- http://charm.cs.uiuc.edu/research/charm/

Add more cache? Has been tried...

Latency via caches

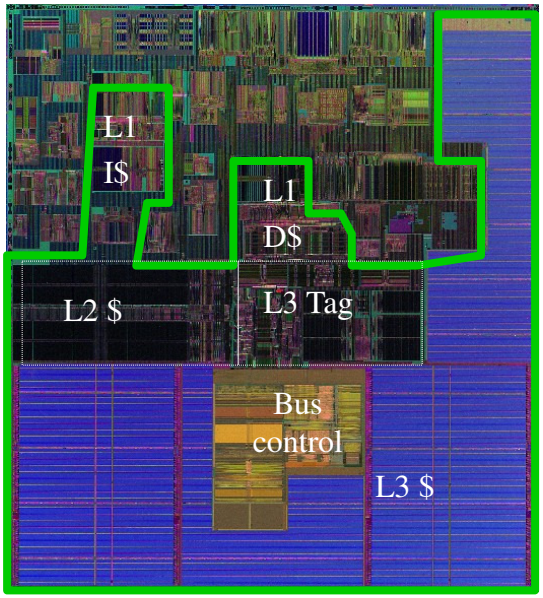
Intel Itanium II has 4 caches on-chip!
- 2 Level 1 caches: 16 KB I and 16 KB D
- Level 2 cache: 256 KB
- Level 3 cache: 3072 KB

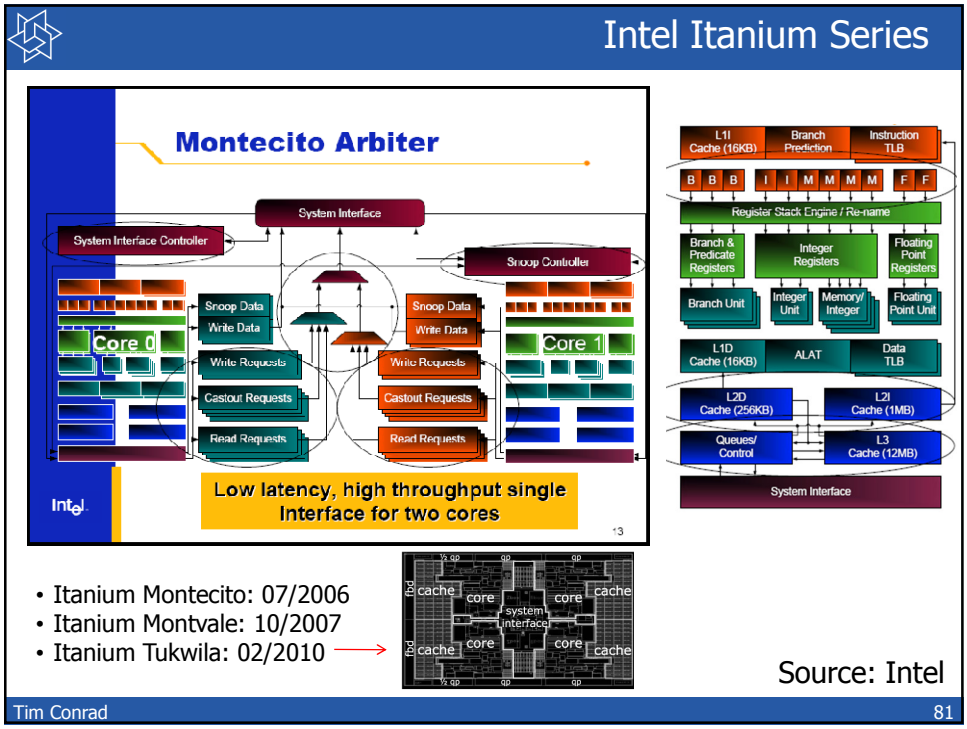
211M transistors
~85% for caches

Die size 421 mm^2

130 Watts @ 1GHz

1% die to change data, 99% to move, store data?

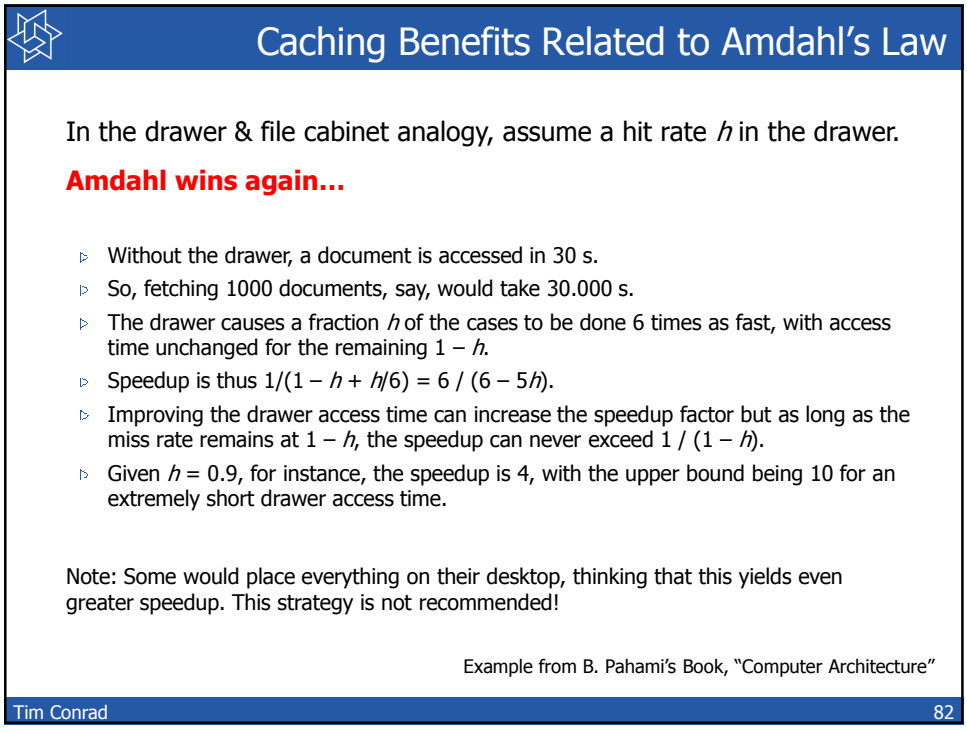




Intel Itanium Series

- Itanium Montecito: 07/2006
- Itanium Montvale: 10/2007
- Itanium Tukwila: 02/2010 →

Source: Intel



Caching Benefits Related to Amdahl's Law

In the drawer & file cabinet analogy, assume a hit rate *h* in the drawer.

Amdahl wins again...

- Without the drawer, a document is accessed in 30 s.
- So, fetching 1000 documents, say, would take 30.000 s.
- The drawer causes a fraction *h* of the cases to be done 6 times as fast, with access time unchanged for the remaining 1 – *h*.
- Speedup is thus $1/(1 - h + h/6) = 6 / (6 - 5h)$.
- Improving the drawer access time can increase the speedup factor but as long as the miss rate remains at 1 – *h*, the speedup can never exceed 1 / (1 – *h*).
- Given $h = 0.9$, for instance, the speedup is 4, with the upper bound being 10 for an extremely short drawer access time.

Note: Some would place everything on their desktop, thinking that this yields even greater speedup. This strategy is not recommended!

Example from B. Pahami's Book, "Computer Architecture"