Why Parallel

the greed for speed is a permanent malady

2 basic options:

- Build a faster uniprocessor
  - advantages
    - programs don't need to change
    - compilers may need to change to take advantage of intra-CPU parallelism
  - disadvantages
    - improved CPU performance is very costly - we already see diminishing returns
    - very large memories are slow
- Parallel Processors
  - today implemented as an ensemble of microprocessors
    - SAN style interconnect
    - large variation in how memory is treated

Parallel Processors

- The high end requires this approach
  - DOE's ASCI program for example
- Advantages
  - leverage off the sweet spot technology
  - huge partially unexplored set of options
- Disadvantages
  - software - optimized balance and change are required
  - overheads - a whole new set of organizational disasters are now possible

Types of Parallelism

Note: many overlaps

- lookahead & pipelining
- vectorization
- concurrency & simultaneity
- data and control parallelism
- partitioning & specialization
- interleaving & overlapping of physical subsystems
- multiplicity & replication
- time & space sharing
- multitasking & multiprogramming
- multi-threading
- distributed computing - for speed or availability

Historical Perspective

Generation	Technology and Architecture	Software and Applications	Representative Systems
First (1945 - 1954)	Vacuum tubes and relay memories - simple PC and ACC	Machine language Single user Programmed I/O	ENIAC Princeton IAS IBM 701
Second (1955 - 1964)	Discrete transistor Core Memory Floating point arith. I/O Processors	Fortran & Cobol Subroutine libraries Batch processing OS	IBM 7090 CDC 1604 Univac LARC Burroughs B5500
Third (1965 - 1974)	SSI and MSI IC's microprogramming pipelining, cache, and loo-kahead	More HLL's Multiprogramming and Timesharing OS Protection and file system capability	IBM 360/370 CDC 6600 TI ASC PDP-88
Fourth (1975 - 1990)	LSI/VLSI processors, semi-conductor memory, vector supercomputers, multicomputers	Multiprocessor OS, parallel languages, multiuser applications	VAX 9000 Cray X-MP FPS T2000 IBM 3090
Fifth (1991 - present)	ULSI/VHSIC processors, memory, switches. High density packages and scalable architectures	MPP, grand challenge applications, distributed and heterogeneous process-ing, I?O becomes real	IBM SP SGI Origin Intel ASCI Red

What changes when you get more than 1?

everything is the easy answer!

2 areas deserve special attention

- Communication
  - 2 aspects always are of concern
    - latency & bandwidth
  - before - I/O meant disk/etc. = slow latency & OK bandwidth
  - now - interprocessor communication = fast latency and high bandwidth - becomes as important as the CPU
- Resource Allocation
  - smart Programmer - programmed
  - smart Compiler - static
  - smart OS - dynamic
  - hybrid - some of all of the above is the likely balance point

Inter-PE Communication

software perspective

- Implicit via memory
  - distinction of local vs. remote
  - implies some *shared* memory
  - sharing model and access model must be consistent
- Explicitly via send and receive
  - need to know destination and what to send
  - blocking vs. non-blocking option
  - usually seen as message passing

Inter-PE Communication

hardware perspective

- Senders and Receivers
  - memory to memory
  - CPU to CPU
  - CPU activated/notified but transaction is memory to memory
  - which memory - registers, caches, main memory
- Efficiency requires
  - consistent SW & HW models
  - policies should not conflict

Communication Performance

critical for MP performance

- 3 key factors
  - bandwidth
    - does the interconnect fabric support the needs of the whole collection
    - scalability issues
  - latency
    - = sender overhead + time of flight + transmission time + receiver overhead
    - transmission time = interconnect overhead
  - latency hiding capability of the processor nodes
    - lots of idle processors is not a good idea

detailed study of interconnects
last chapter topic
since we need to understand I/O first

Flynn's Taxonomy - 1972

too simple but it's the only one that moderately works
4 Categories = (Single, Multiple) X (Data Stream, Instruction Stream)

- SISD - conventional uniprocessor system
  - still lots of intra-CPU parallelism options
- SIMD - vector and array style computers
  - started with ILLIAC
  - first accepted multiple PE style systems
  - now has fallen behind MIMD option
- MISD - ~ systolic or stream machines
  - example: iWarp and MPEG encoder
- MIMD - intrinsic parallel computers
  - lots of options - today's winner - our focus

MIMD options

- Heterogeneous vs. Homogeneous PE's
- Communication Model
  - explicit: message passing
  - implicit: shared-memory
  - oddball: some shared some non-shared memory partitions
- Interconnection Topology
  - which PE gets to talk directly to which PE
  - blocking vs. non-blocking
  - packet vs. circuit switched
  - wormhole vs. store and forward
  - combining vs. not
  - synchronous vs. asynchronous

The Easy and Cheap Obvious Option

- Microprocessors are cheap
- Memory chips are cheap
- Hook them up somehow to get n PE's
- Multiply each PE's performance by n and get an impressive number

What's wrong with this picture?

- most uP's have been architected to be the only one in the system
- most memories only have one port
- interconnect is not just somehow
- anybody who computes system performance with a single multiply is a moron

Ideal Performance - the Holy Grail

- Requires perfect match between HW & SW
- Tough given static HW and dynamic SW
  - hard means cast in concrete
  - soft means the programmer can write anything
- Hence performance depends on:
  - The hardware: ISA, memory, cycle time, etc.
  - The software: OS, task-switch, compiler, application code
- Simple performance model (aka uniprocessor)

$$\text{CPU-time (T)} = \text{Instruction-count (Ic)} \times \text{CPI} \times \text{Cycle-time}(\tau)$$

- But CPI can vary by more than 10x

CPI Stretch Factors

- Conventional Uniprocessor factors
  - TLB miss penalty, page fault penalty, cache miss penalty
  - pipeline stall penalty, OS fraction penalty
- Additional Multiprocessor Factors
  - shared memory
    - non-local access penalty
    - consistency maintenance penalty
  - message passing
    - Send penalty even for non-blocking
    - Receive or notification penalty - task switch penalty (probably 2x)
    - Body copy penalty
    - Protection check penalty
    - Etc. - the OS fraction goes up typically

The Idle Factor Paradox

- After the stretch factor - the performance equation becomes

$$T = Ic \times CPI \times stretch \times \tau$$

- For an ideally scalable n PE system T/n will be the CPU time required
- But idle time will create it's own penalty
- Hence

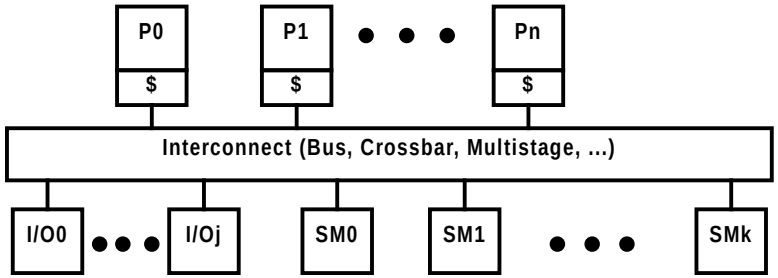
$$T = \frac{\sum_{i=1}^{n} \frac{Ic \times CPI \times stretch \times \tau}{(1 - \%idle)}}{n}$$

- What if %idle goes up faster than n?

Shared Memory UMA

Uniform Memory Access

- Sequent Symmetry S-81
  - symmetric ==> all PE's have same access to I/O, memory, executive (OS) capability etc.
  - asymmetric ==> capability at PE's differs



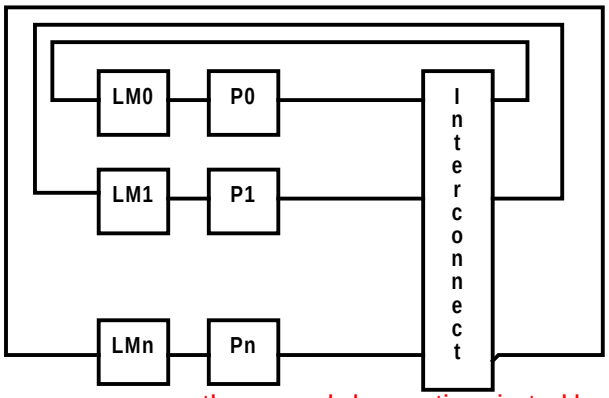
Modern NUMA View

- All uP's set up for SMP
  - SMP ::= symmetric multiprocessor
    - communication is usually the front side bus
  - example
    - Pentium III and 4 Xeon's set up to support 2 way SMP
    - just tie the FSB wires
  - as clock speeds have gone up for n-way SMP's
    - FSB capacitance has reduced the value of n
- Chip based SMP's
  - IBM's Power 4
    - 2 Power 3 cores on the same die
    - set up to support 4 cores

NUMA Shared Memory opus 1 level

Non-Uniform Memory Access

- BBN Butterfly + others



NOTES:
transfer initiated by: LMx or Px

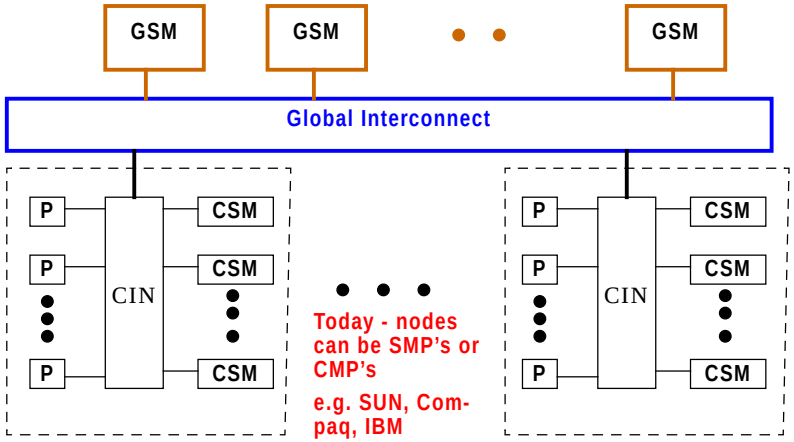
Answer to:
LMx or Px

All options have been seen in practice

the easy and cheap option - just add interconnect

NUMA Shared Memory opus 2 level

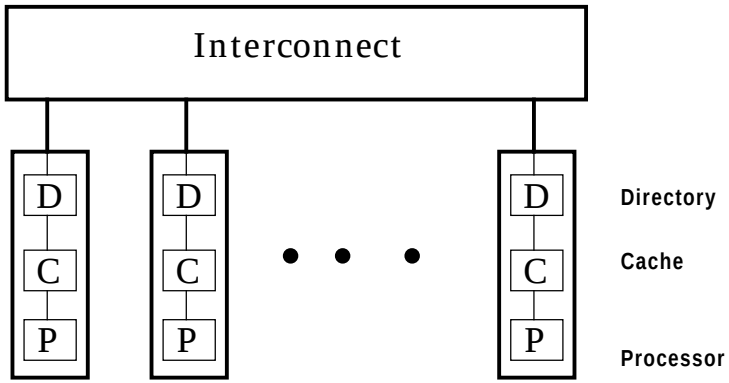
- e.g. Univ. of Ill. Cedar + CMU CM* & C.mmp



COMA Shared Memory

Cache Only Memory Access

- e.g. KSR-1



Lots of other DSM variants

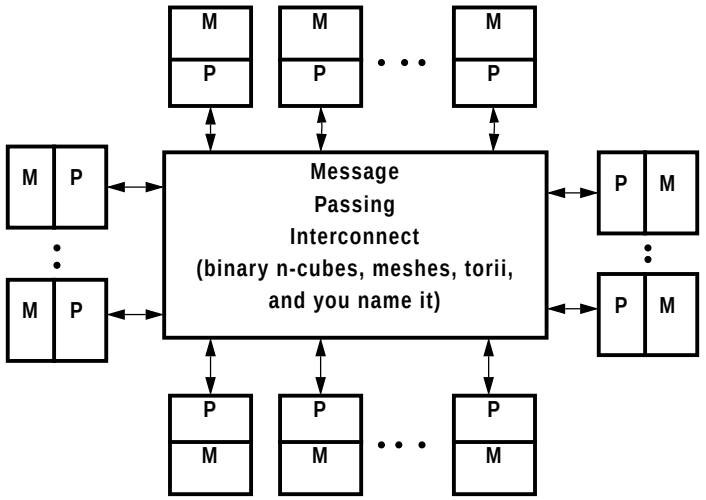
- Cache consistency
  - DEC Firefly - up to 16 snooping caches in a workstation
- Directory based consistency
  - like the COMA model but deeper memory hierarchy
  - e.g. Stanford DASH machine, MIT Alewife, Alliant FX-8
- Delayed consistency
  - many models for the delayed updates
  - a software protocol more than a hardware model
    - e.g. MUNIN - John Carter (good old U of U)
  - other models - Alan Karp and the IBM crew

NORMA

No remote memory access = message passing

- Many players:
  - Schlumberger FAIM-1
  - HPL Mayfly
  - CalTech Cosmic Cube and Mosaic
  - NCUBE
  - Intel iPSC
  - Parsys SuperNode1000
  - Intel Paragon
- Remember the simple and cheap option?
  - with the exception of the interconnect
  - this is the simple and cheap option

Message Passing MIMD Machines



Message Passing vs. Shared Memory

- shared memory advantages
  - programming model is simple and familiar
  - quick port of existing code - then try to parallelize but at least something is running that you can profile
  - low communication overhead for small items
  - OS isn't in the way of a memory reference
  - cacheing helps alleviate communication needs
- message passing advantages
  - simple hardware ==> faster
  - communication is explicit - good and bad news for the programmer
  - natural synchronization - associated with messages
- duality
  - either model can be built on the other
  - easier to map message passing onto shared memory than vice versa
  - funny: message passing on the Origin 2K was faster than on the IBM SP2 (note this hopefully has changed)

Parallel Performance Challenge

- Amdahl's law in action
  - enhanced = parallel in this case
  - example1 - code centric
    - 80% of your code is parallel
    - ==> best you can do is get a speed up of 5 no matter how many processors you throw at the problem
  - example 2 - speedup centric
    - want 80x speedup on 100 processors
    - ==> $\text{fraction}_{\text{parallel}} = .9975$
    - this will be hard
- Linear speed up is hard
- Superlinear speed up is easier
  - lots more memory may remove the need to page

Modern Remote Memory Access Times

critical limit for shared memory performance

Multiprocessor	Year Shipped	Type	max PE count	Interconnect	Remote Memory Access (ns)
Sun Starfire Servers	1996	SMP	64	multiple buses	500
SGI Origin 3000	1999	NUMA	512	fat hypercube	500
Cray T3E	1996	NUMA	2048	2-way 3D torus	300
HP V series	1998	SMP	32	8x8 crossbar	1000
Compaq AlphaServer GS	1999	SMP	32	switched busses	400

Parallel Workloads

- Even more disparate
  - application characterists
  - performance varies with
    - uniprocessor and communication utilization
    - wide variance with architecture type
- 3 workloads studied
  - commercial
    - OLTP based on TPC-B
    - DSS based on TPC-D
    - Web index search based on AltaVista and a 200GB database
  - multiprogrammed & OS
    - 2 independent copies of compiling the Andrew file system
    - phases: compile (compute bound), install object files, remove files (I/O bound)
  - Scientific/Technical
    - FFT, LU, Ocean, and Barnes

Workload Effort Characteristics

Commercial Workload (4 processor AlphaServer 4100)

Benchmark	% time in user mode	% time in kernal mode	% time CPU idle
OLTP	71	18	11
DSS range for all 6 Queries	82-94	3-5	4-13
DSS average	87	3.7	9.3
AltaVista	>98	<1	<1

Multiprogrammed & OS (8 processors - simulated)

	User	Kernel	Synch Wait	CPU idle (I/O wait)
% instructions xeq'd	27	3	1	69
% xeq time	27	7	2	64

Scientific/Technical

- FFT
  - 1D version for a complex number FFT
    - 3 data structures - in and out arrays plus a precomputed read-only roots matrix
  - steps
    - transpose the data matrix
    - 1D FFT on each row of data
    - multiply roots matrix by the data matrix
    - transpose data matrix
    - 1D FFT on each row of data matrix
    - transpose data matrix
  - communication
    - all to all communication in the three transpose phases
    - each processor transposes one block locally and sends one block to each other processor

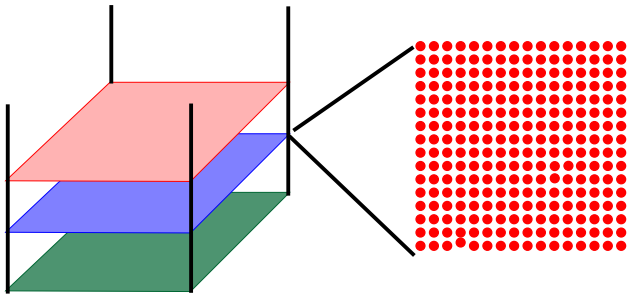
The Other Kernel

- LU
  - typical dense matrix factorization
    - used in a variety of solvers and eigenvalue computations
  - turn a matrix into a upper diagonal
    - blocking helps code to be cache friendly
  - block size
    - small enough to keep cache miss rate low
    - large enough to maximize the parallel phase

Ocean

- Goal
  - global weather modeling
  - note that 75% of the earth's surface is ocean
    - ocean currents and atmosphere have a major weather impact
    - near vertical walls there is a significant *eddy* effect
- Physical problem
  - continuous in both 1D time and 3D space
- Discrete model for simulation
  - model the ocean as a discrete set of points equally spaced
    - point variables for pressure, current direction and speed, temperature, etc.
  - simplify here to a set of 2D point planes
    - admittedly less accurate and changes convergence aspects
    - eases the use of this application and still points out key issues

Ocean's Ocean Model



Rectangular basin = 3D

simplify = 2d plane set

separate 2d array for each variable

equal spaced points

continuous ==> discrete

Ocean

the benchmark

- Data
  - 2D arrays for each variable
  - all arrays model each cross section plane
- Time
  - solving system of motion equations
  - sweep through all of the points for some point in time
  - continue to next time step
- Granularity
  - big influence on computation time
    - 2Mm x 2Mm = atlantic ocean
    - 5 years of 1 minute time steps & 1 Km spacing = 2.628Msteps for 4 x 10^6 pts is intractable
    - must go to larger grain for now - however solution style is the key here

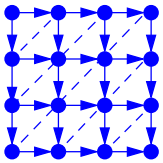
Ocean

equation kernel solver

- Solves a differential equation
  - via a finite difference method
  - operates on a (n+2) x (n+2) matrix
    - +2 ==> border rows and columns which do not change while interior is being computed.
    - then borders are changed and communicated and then we go to the next step
  - uses Gauss-Seidel update
    - computes a weighted average for each point based on 4 neighbors
    - order may vary but let's start with row-major order
    - implies new values from above and left but old values from below and right neighbors
  - step termination
    - if the sum of the difference for all points is less than some tolerance then done, otherwise make another sweep over the array

Decomposition

- Model the weighted nearest neighbor average
  - A[i,j] = 0.2 x (A[i,j] + A[i,j-1] + A[i-1,j] + A[i,j+1] + A[i+1,j]

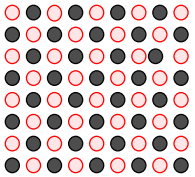


Evolve the sequential algorithm

bogus once again - little parallelism

Note the anti-diagonal option (orthogonal to resultant dependence vector)

Control and Load Imbalance Issues??



Red Black Decomposition

Dependencies?

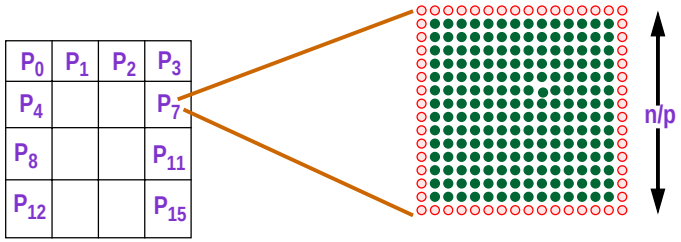
Parallelism?

Convergence properties?

Ocean Communication

side effect of blocked grid based solver

perimeter vs. area

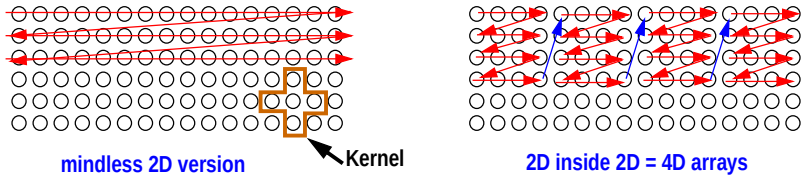


Local Work $\alpha \frac{n^2}{p}$

Remote Communication $\alpha \frac{4n}{\sqrt{p}}$

Blocking in Ocean

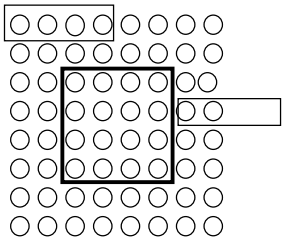
influences cache locality



- consider cache effects
  - spatial and temporal locality
- other effects
  - blocks can also be influenced by processor partition
  - particularly useful if address space is shared as in a DSM machine
- boundary problems?

Boundary Issues

- Inherent problem
  - assume row major order
  - column lines will have poor spatial locality



Ocean

- Sequential algorithm
  - outer loop over a very large number of time steps
  - time-step = 33 computations
    - each one using a small number of grid variables
  - typical computation
    - sum of scalar multiples from close by grid points
    - nearest neighbor averaging sweep
  - add multigrid technique
    - levels of coarseness: +1 ==> ignore every other grid
    - start at finest level and look at the diff
    - if small but above tolerance then bump up +1 coarser
    - if large then -1 coarseness (in between then stay at this level)
    - accelerates convergence

Barnes-Hut

- Simulates evolution of galaxies
  - classic N-body problem
- Characteristics
  - no spatial regularity so computation becomes particle based
  - every particle exerts influence on every other particle
    - ugh - hence $O(n^2)$
    - but clustering of distant star groups can be based on center of mass since

$$\text{Gravitational Force} = G\frac{M_1 M_2}{r^2}$$

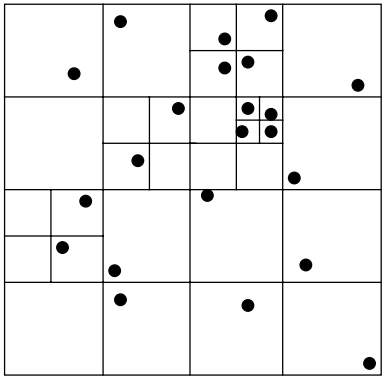
    - result is O(n log n)
    - close stars must be represented individually

Octtree Hierarchy

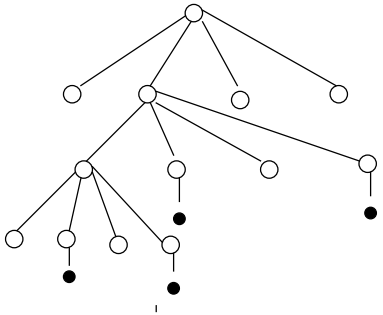
- 3D galaxy represented as a octree
  - divide galaxy recursively into 8 equally sized children
    - based on equal space volumes independent of membership
    - if a subspace has more than x bodies then subdivide again
    - x typically will be something like 8
  - tree is traversed once per body
    - determines force on that body
  - bodies move so tree is rebuilt every time step
- Group optimization
  - if the cell is far enough away
    - l/d < x, l=length of a side of the cell, d distance of body from cell center of mass, and x is the accuracy parameter (typically between .5 and 1.2)
  - then treat the cell as a single body
  - otherwise you have to open the cell and proceed

Tree Example

2D = quadtree



2D Spatial Decomposition



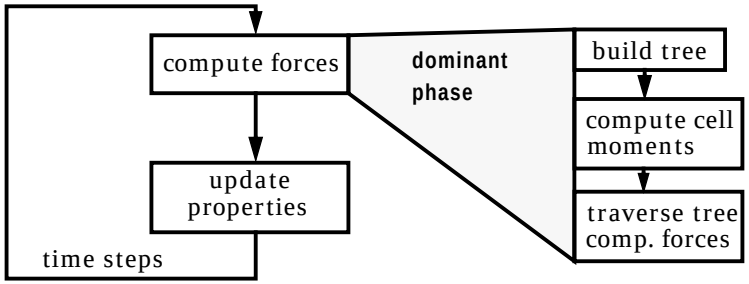
QuadTree Equivalent

Each non-leaf has center of mass for the group

Each leaf has mass, velocity, etc.

Barnes-Hut

- Sequential Algorithm
  - 100's of time steps
  - each step computes the net force on every body
    - updates body position and other attributes (velocity, acceleration, direction)
  - flow



S/T Workload Scaling

Application	Computation Scaling per processor	Communication Scaling	Compute/ Communicate Scaling
FFT	(n log n)/p	n/p	log n
LU	n/p	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$
Barnes	(n log n)/p	approximately $\frac{\sqrt{n}\log n}{\sqrt{p}}$	approximately $\frac{\sqrt{n}}{\sqrt{p}}$
Ocean	n/p	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$