CS 267 Applications of Parallel Computers

Lecture 28:
LogP and the
Implementation and Modeling of Parallel Sorts

James Demmel

(taken from David Culler,

Lecture 18, CS267, 1997)

http://www.cs.berkeley.edu/~demmel/cs267_Spr99

© Culler 1997

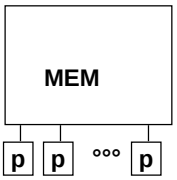
Practical Performance Target (circa 1992)

- Sort one billion large keys in one minute on one thousand processors.
- Good sort on a workstation can do 1 million keys in about 10 seconds
  - just fits in memory
  - 16 bit Radix Sort
- Performance unit: **µs per key per processor**
  - s ~ 10 for single Sparc 2

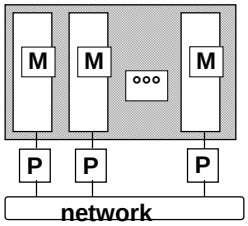
© Culler 1997

Studies on Parallel Sorting

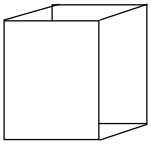
PRAM Sorts



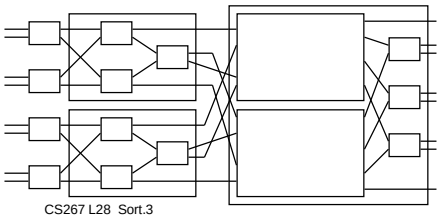
LogP Sorts



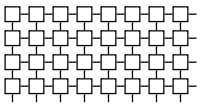
Sorting on Machine X



Sorting Networks

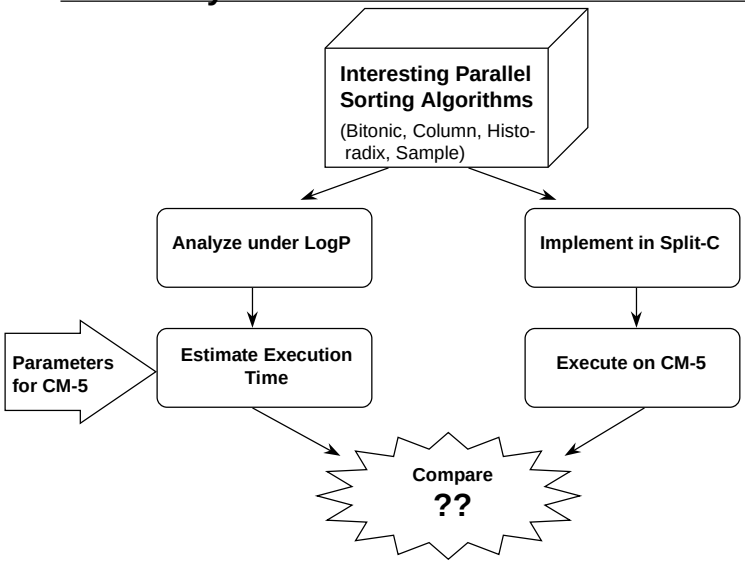


Sorting on Network Y



© Culler 1997

The Study



© Culler 1997

LogP

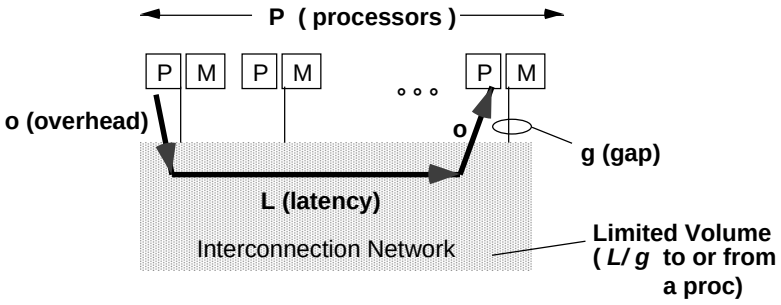
© Culler 1997

Deriving the LogP Model

° Processing
  – powerful microprocessor, large DRAM, cache => P
° Communication
  + significant latency (100's of cycles) => L
  + limited bandwidth (1 – 5% of memory bw) => g
  + significant overhead (10's – 100's of cycles) => o
    - on both ends
  – no consensus on topology
    => should not exploit structure
  + limited capacity
  – no consensus on programming model
    => should not enforce one

© Culler 1997

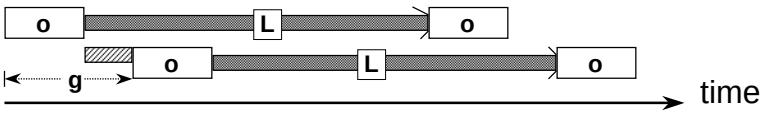
LogP



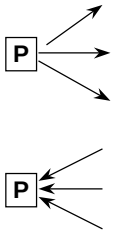
- Latency in sending a (small) mesage between modules
- overhead felt by the processor on sending or receiving msg
- gap between successive sends or receives (1/BW)
- Processors

© Culler 1997

Using the Model



° Send n messages from proc to proc in time *2o + L + g(n-1)*
  – each processor does *o n* cycles of overhead
  – has *(g-o)(n-1) + L* available compute cycles
° Send *n* messages from *one to many* in same time
° Send *n* messages from *many to one* in same time
  – all but *L/g* processors block so fewer available cycles



© Culler 1997

Use of the Model (cont)

° **Two processors sending *n* words to each other (**i.e., exchange**) in time**

2o + L + max(g,2o) (n-1) ≤ max(g,2o) + L

° ***P*** **processors each sending *n* words to all processors (*n/P* each) in a static, balanced pattern without conflicts ,** e.g., transpose, fft, cyclic-to-block, block-to-cyclic

same

exercise: what's wrong with the formula above?

Assumes optimal pattern of send/receive, so could underestimate time

© Culler 1997

LogP "philosophy"

- Think about:
- – mapping of N words onto P processors
- – computation within a processor, its cost, and balance
- – communication between processors, its cost, and balance
- given a charaterization of processor and network performance

- Do not think about what happens *within* the network

This should be good enough!

© Culler 1997

Typical Sort

Exploits the n = N/P grouping

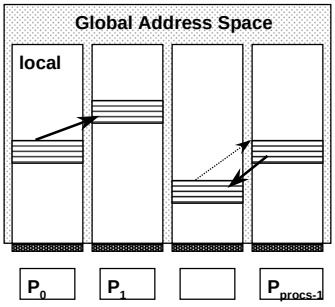
° Significant local computation

° Very general global communication / transformation

° Computation of the transformation

© Culler 1997

Split-C



- Explicitly parallel C
- 2D global address space
  - linear ordering on local spaces
- Local and Global pointers
  - spread arrays too
- Read/Write
- Get/Put (overap compute and comm)
  - x := G; . . .
  - sync();
- Signaling store (one-way)
  - G :– x; . . .
  - store_sync(); or all_store_sync();
- Bulk transfer
- Global comm.

© Culler 1997

Basic Costs of operations in Split-C

• Read, Write	x = *G, *G = x	2 (L + 2o)
• Store	*G :– x	L + 2o
• Get	x := *G	o
	....	2L + 2o
	sync();	o

  – with interval g

- Bulk store (n words with ω words/message)

  2o + (n-1)g + L
- Exchange $2o + 2L + (\lceil n/\omega \rceil - 1 - L/g)\ \max(g,2o)$

- One to many
- Many to one

© Culler 1997

LogP model

- CM5:
  – L = 6 µs
  – o = 2.2 µs
  – g = 4 µs
  – P varies from 32 to 1024
- NOW
  – L = 8.9
  – o = 3.8
  – g = 12.8
  – P varies up to 100

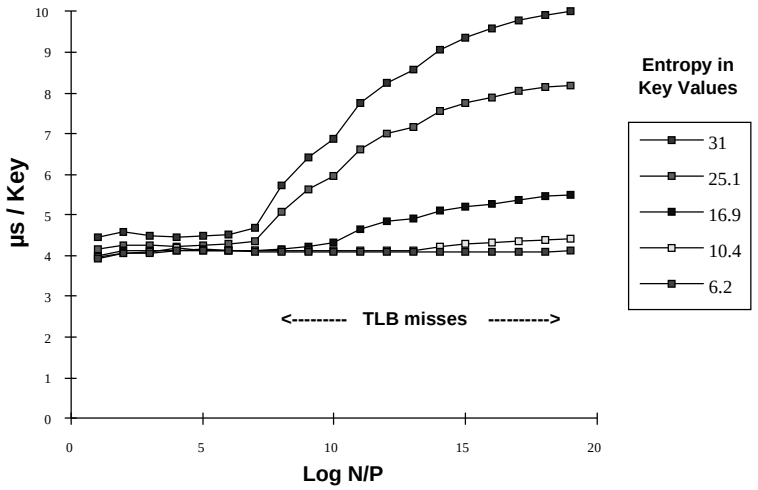
- What is the processor performance?

© Culler 1997

Sorting

© Culler 1997

Local Sort Performance (11 bit radix sort of 32 bits numbers)



Entropy = $-\Sigma_i p_i \log p_i$,
p_i = Probability of key i

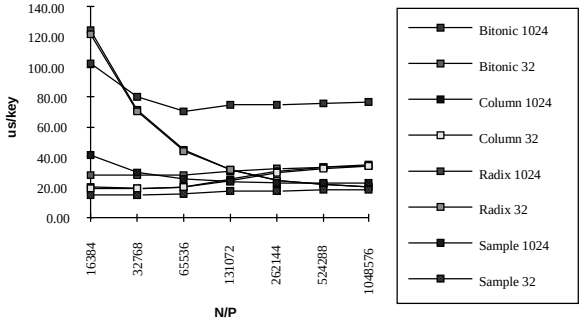
© Culler 1997

Local Computation Parameters - Empirical

Parameter	Operation	µs per key	Sort
Swap	Simulate cycle butterfly per key	0.025 lg N	Bitonic
mergesort	Sort bitonic sequence	1.0	
scatter	Move key for Cyclic-to-block	0.46	
gather	Move key for Block-to-cyclic	0.52 if n<=64k or P<=64	Bitonic & Column
		1.1 otherwise	
local sort	Local radix sort (11 bit)	4.5 if n < 64K	
		9.0 - (281000/n)	
merge	Merge sorted lists	1.5	Column
copy	Shift Key	0.5	
zero	Clear histogram bin	0.2	Radix
hist	produce histogram	1.2	
add	produce scan value	1.0	
bsum	adjust scan of bins	2.5	
address	determine desitination	4.7	
compare	compare key to splitter	0.9	Sample
localsort8	local radix sort of samples	5.0	

© Culler 1997

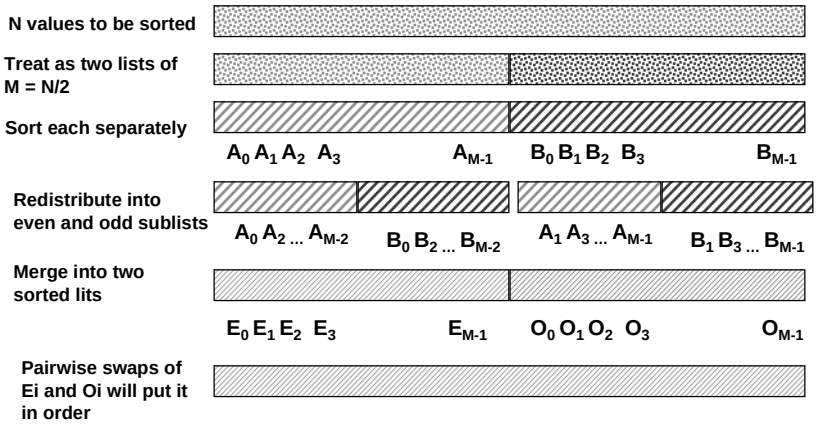
Bottom Line (Preview)



- Good fit between predicted and measured (10%)
- Different sorts for different sorts
  - scaling by processor, input size, sensitivity
- All are global / local hybrids
  - the local part is hard to implement and model

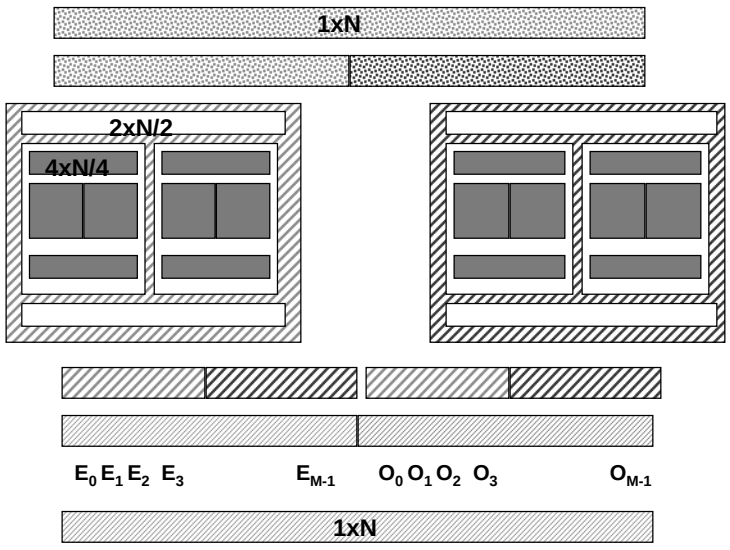
© Culler 1997

Odd-Even Merge - classic parallel sort



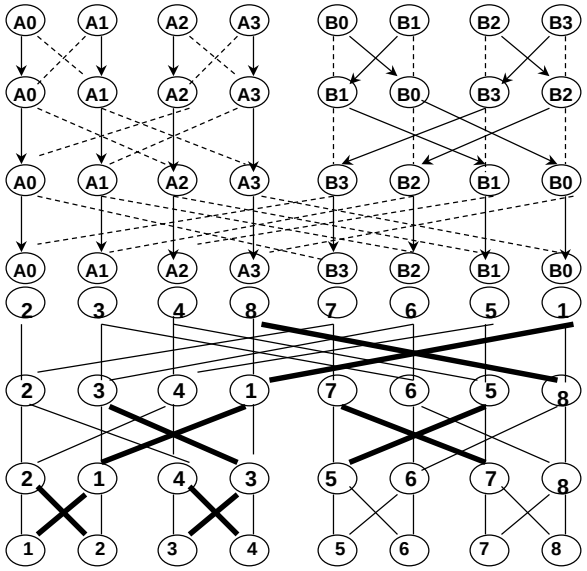
© Culler 1997

Where's the Parallelism?



© Culler 1997

Mapping to a Butterfly (or Hypercube)



two sorted sublists

Reverse Order of one list via cross edges

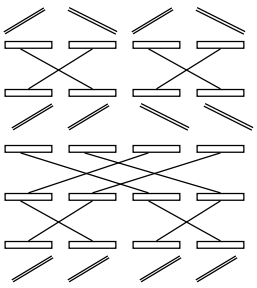
Pairwise swaps on way back

© Culler 1997

Bitonic Sort with N/P per node

A bitonic sequence decreases and then increases (or vice versa)
Bitonic sequences can be merged like monotonic sequences

```
all_bitonic(int A[PROCS]::[n])
   sort(tolocal(&A[ME][0]),n,0)
   for (d = 1; d <= logProcs; d++)
      for (i = d-1; i >= 0; i--) {
         swap(A,T,n,pair(i));
         merge(A,T,n,mode(d,i));
      }
   sort(tolocal(&A[ME][0]),n,mask(i));
```

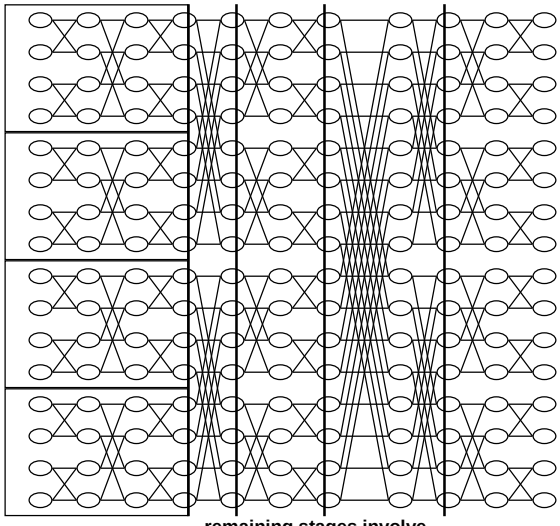


sort
swap

© Culler 1997

Bitonic Sort

Block Layout



lg N/p stages are local sort

remaining stages involve

Block-to-cyclic, local merges (i - lg N/P cols)

cyclic-to-block, local merges (lg N/p cols within stage)

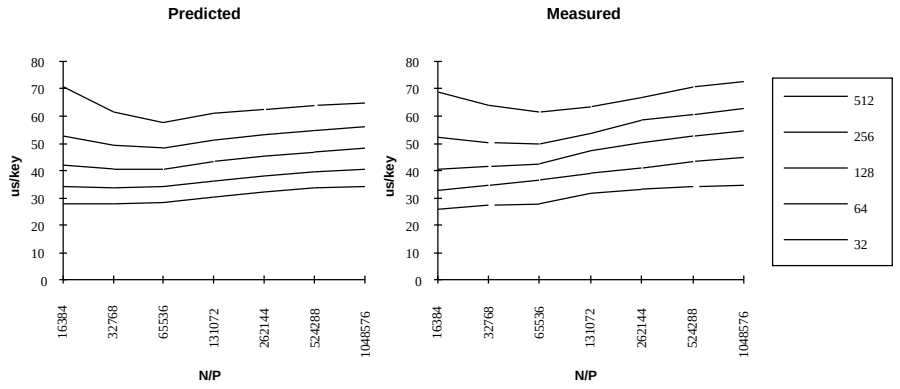
© Culler 1997

Analysis of Bitonic

- How do you do transpose?
- Reading Exercise

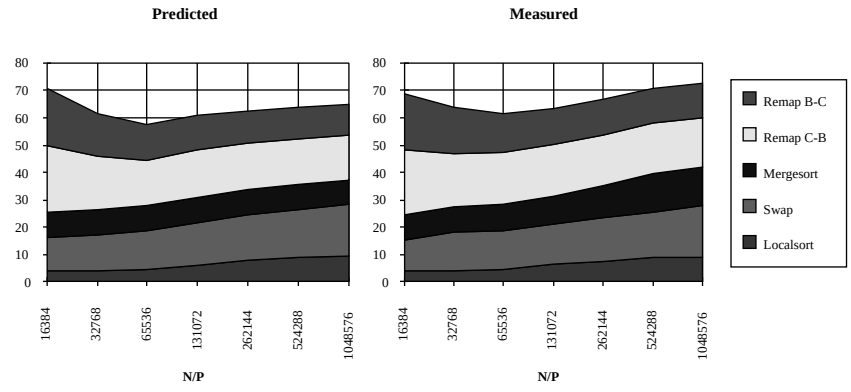
© Culler 1997

Bitonic Sort: time per key



© Culler 1997

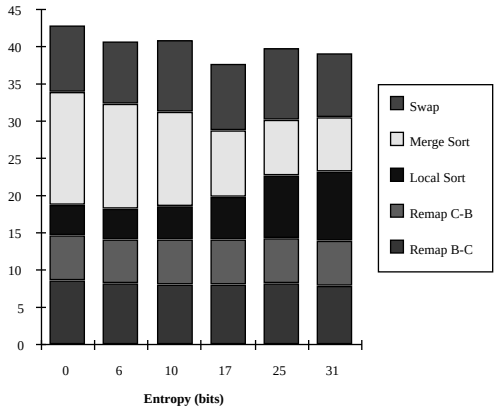
Bitonic: Breakdown



P= 512, random

© Culler 1997

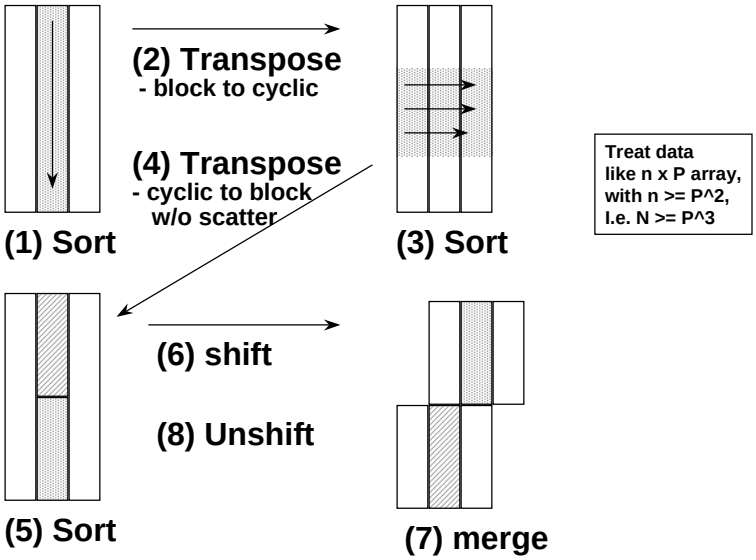
Bitonic: Effect of Key Distributions



P = 64, N/P = 1 M

© Culler 1997

Column Sort



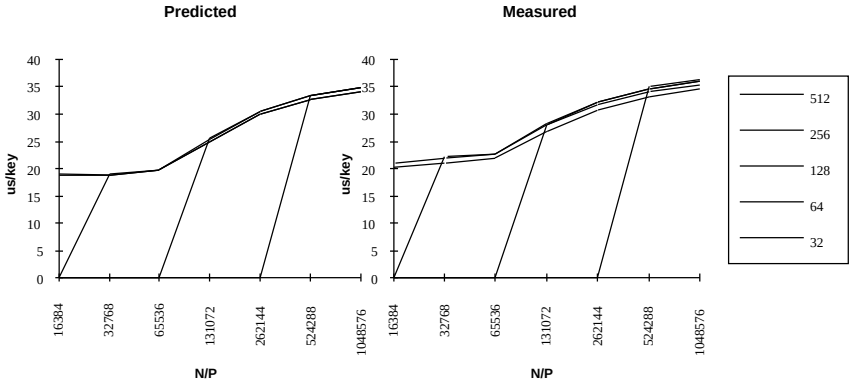
Treat data like n x P array, with n >= P^2, I.e. N >= P^3

work efficient

© Culler 1997

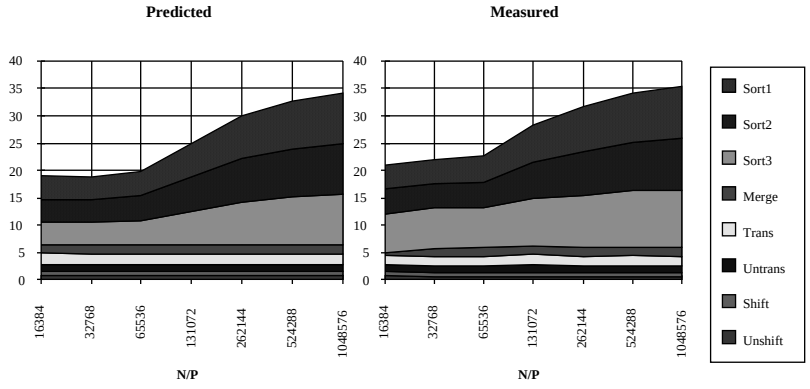
Column Sort: Times

Only works for N >= P^3



© Culler 1997

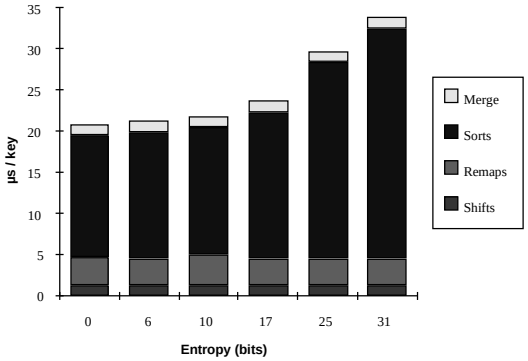
Column: Breakdown



P= 64, random

© Culler 1997

Column: Key distributions



P = 64, N/P = 1M

© Culler 1997

Histo-radix sort

Per pass:

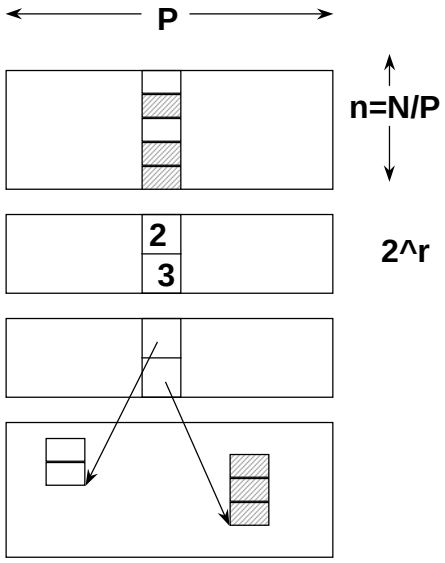
1. compute local histogram

2. compute position of 1st member of each bucket in global array

- **2^r scans with end-around**

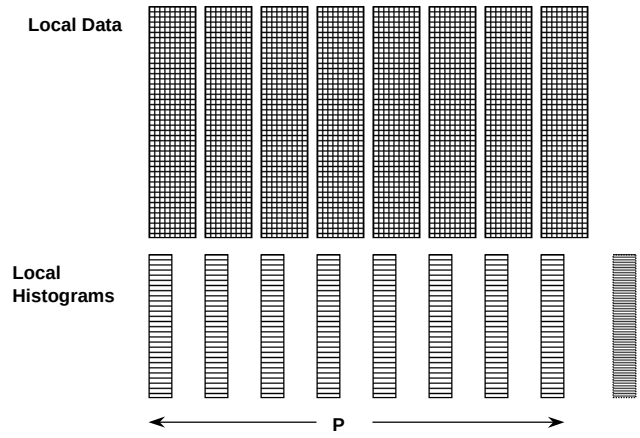
3. distribute all the keys

Only r = 8,11,16 make sense for sorting 32 bit numbers



© Culler 1997

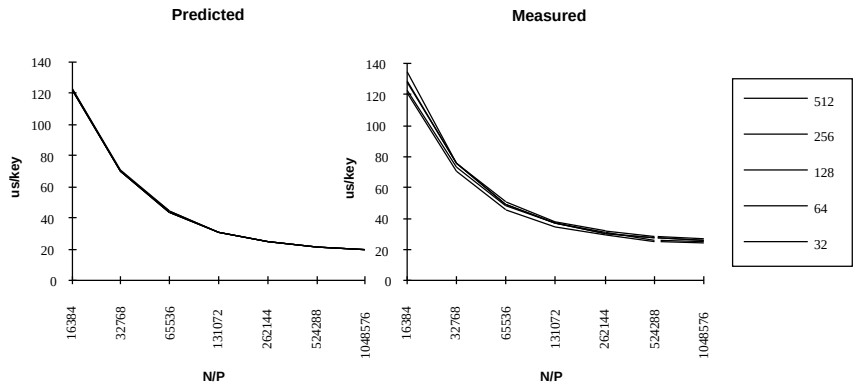
Histo-Radix Sort (again)



Each Pass
- **form local histograms**
- **form global histogram**
- **globally distribute data**

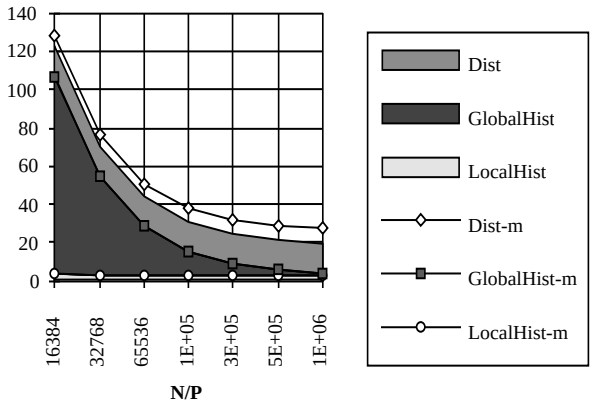
© Culler 1997

Radix Sort: Times



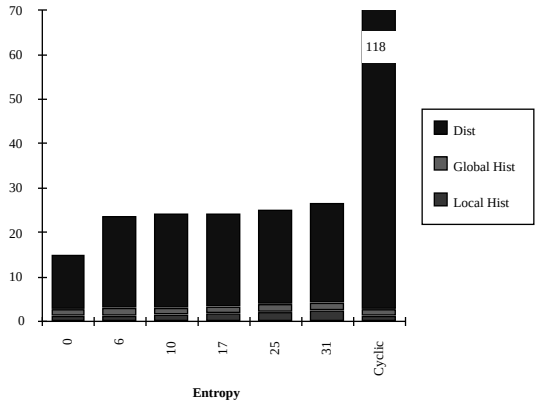
© Culler 1997

Radix: Breakdown



© Culler 1997

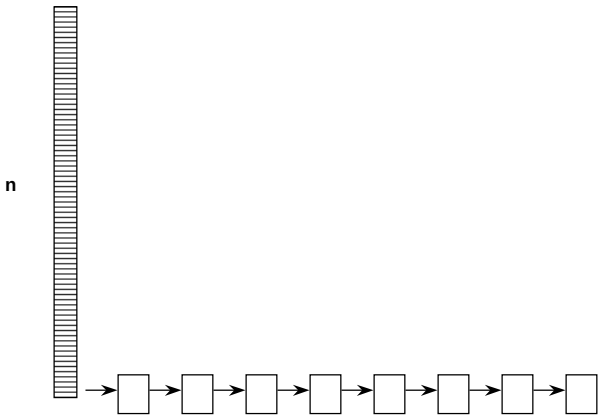
Radix: Key distribution



Slowdown due to contention in redistribution

© Culler 1997

Radix: Stream Broadcast Problem



(P-1) (2o + L + (n-1) g) ? **Need to slow first processor to pipeline well**

© Culler 1997

What's the right communication mechanism?

- Permutation via writes
  - consistency model?
  - false sharing?
- Reads?
- Bulk Transfers?
  - what do you need to change in the algorithm?
- Network scheduling?

© Culler 1997

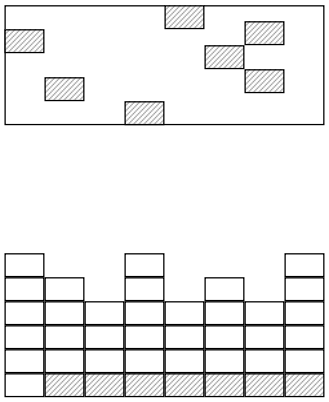
Sample Sort

1. compute P-1 values of keys that would split the input into roughly equal pieces.
- **take S~64 samples per processor**
- **sort PS keys**
- **take key S, 2S, . . . (P-1)S**
- **broadcast splitters**

2. Distribute keys based on splitters

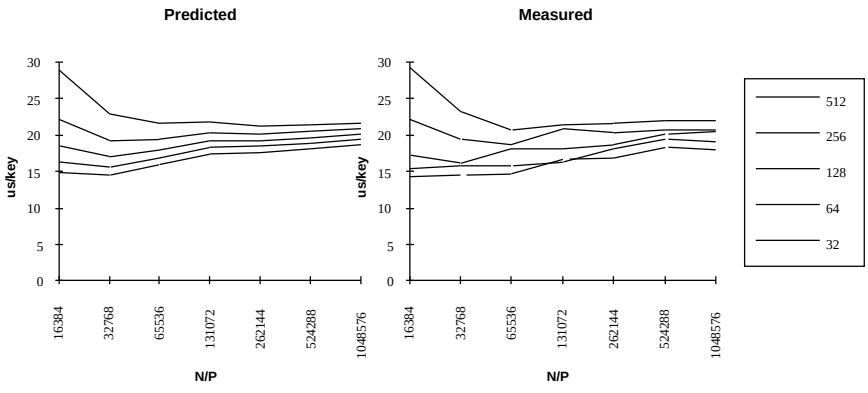
3. Local sort

[4.] possibly reshift



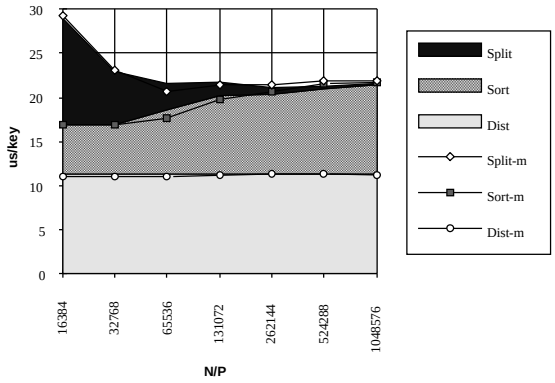
© Culler 1997

Sample Sort: Times



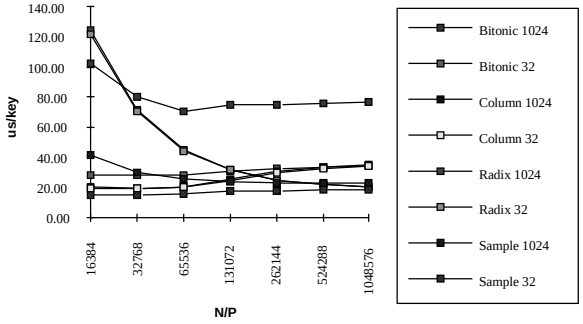
© Culler 1997

Sample Breakdown



© Culler 1997

Comparison



- Good fit between predicted and measured (10%)
- Different sorts for different sorts
  - scaling by processor, input size, sensitivity
- All are global / local hybrids
  - the local part is hard to implement and model

© Culler 1997

Conclusions

- Distributed memory model leads to hybrid global / local algorithms
- LogP model is good enough for the global part
  - bandwidth (g) or overhead (o) matter most
  - including end-point contention
  - latency (L) only matters when BW doesn't
  - g is going to be what really matters in the days ahead (NOW)
- Local computational performance is hard!
  - dominated by effects of storage hierarchy (TLBs)
  - getting trickier with multilevels
    - » physical address determines L2 cache behavior
  - and with real computers at the nodes (VM)
  - and with variations in model
    - » cycle time, caches, . . .
- See http://www.cs.berkeley.edu/~culler/papers/sort.ps
- See http://now.cs.berkeley.edu/Papers2/Postscript/spdt98.ps
  - disk-to-disk parallel sorting

© Culler 1997