LogP Performance Model and Parallel Sorting

Outline

• The Sorting Challenge
• The LogP model
• Modern Sorting Algorithms
  • Radix sort
  • Bitonic sort
  • Sample Sort

Adapted from D. Culler and Yelick’s talks
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: $\mu$s per key per processor
  - $s \sim 10$ for single Sparc 2

Adapted from D. Culler and Yelick's talks

Performance Targets in 2001 (Jim Gray)

- Datamation (1 April, 1985)
  - Sort a million hundred-byte records
  - tests file system, IO system, utility access

- Minute Sort:
  - Sort as many records as you can in a minute
  - Report rate and price ($\text{cost}/1\text{e}6$

- Penny Sort:
  - Sort as much as you can for a penny.
  - Two versions: Daytona (stock car) and Indy (formula 1)
  - Proposed Change to PennySort: Performance/Price Sort
  - Compute GB/$ of a two pass sort: performance per dollar.

Adapted from D. Culler and Yelick's talks
Datamation Results

- Was one hour, now less ½ second

Studies on Parallel Sorting

PRAM Sorts

LogP Sorts

Sorting on Machine X

Sorting on Network Y

Adapted from D. Culler and Yelick's talks
The Study

Analyze under LogP
Parameters for machine

Estimate Execution Time

Implement in Split-C

Execute on CM-5

Compare

(Bitonic, Column, Histo-radix, Sample)

Adapted from D. Culler and Yelick's talks

LogP Performance Model in Parallel Machines

Adapted from D. Culler and Yelick's talks
**Deriving the LogP Model**

° Processing
  – powerful microprocessor, large DRAM, cache  => P

° Communication
  + significant latency  (100’s – 1000’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 network capacity
    – no consensus on programming model
    => should not enforce one

---

**LogP**

- Latency in sending a (small) message between modules  \( \alpha \)
- Overhead felt by the processor on sending or receiving msg  \( \beta \)
- gap between successive sends or receives (1/BW)
- Processors

Adapted from D. Culler and Yelick's talks
Using the LogP Model

° Send \( n \) messages from proc to proc in time
\[ 2o + L + g \ (n-1) \]
– each processor does \( o \cdot n \) cycles of overhead
– has \((g-o)(n-1) + L\) available compute cycles
° Send \( n \) total 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, unless scheduled carefully

Adapted from D. Culler and Yelick’s talks

Use of the LogP Model (cont)

° Two processors sending \( n \) words to each other (i.e., exchange)
\[ 2o + L + \max(g,2o) \ (n-1) \leq \max(g,2o) + L \]

° \( P \) 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

exercise: what’s wrong with the formula above?
Assumes optimal pattern of send/receive, so could underestimate time

Adapted from D. Culler and Yelick’s talks
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 characterization of processor and network performance

- Do not think about what happens within the network

This should be good enough!

Typical Sort

Exploits the n = N/P grouping

- Significant local computation
- Very general global communication / transformation
- Computation of the transformation

Adapted from D. Culler and Yelick's talks
Split-C: Predecessor to UPC

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

Get/put similar to “relaxed”
Signaling store not in UPC

Basic Costs of operations in Split-C

- Read, Write \( x = *G, *G = x \) 2 (L + 2o)
- Store \( *G := x \) L + 2o
- Get \( x := *G \) 0
  .... 2L + 2o
  sync(); 0

  - with interval g
- Bulk store (n words with \( \omega \) 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

Adapted from D. Culler and Yelick's talks
LogP model

- **CM5:**
  - $L = 6 \mu s$
  - $o = 2.2 \mu s$
  - $g = 4 \mu s$
  - $P$ varies from 32 to 1024
- **NOW: Network of Workstation**
  - $L = 8.9$
  - $o = 3.8$
  - $g = 12.8$
  - $P$ varies up to 100

- What is the processor performance?
  - Application-specific
  - 10s of Mflops for these machines

Adapted from D. Culler and Yelick's talks

LogP Parameters Today

- Millennium (MPICH/M2K)
- SP Loopback (MPI)
- SP (MPI)
- Quadrics loopback (MPI, ORNL)
- Quadrics (MPI, ORNL)
- Quadrics (Shmem, ORNL)
- Quadrics (UPC, ORNL)
- Myrinet (GM)
- MVICH (Gigabit)
- VIPL (VIA)
- MVICH/M-VIA/Syskonnect
- VIPL/M-VIA/Syskonnect
- T3E Loopback (MPI, NERSC)
- T3E (MPI, NERSC)
- T3E UPC (NERSC)

Adapted from D. Culler and Yelick's talks
Parallel Sorting

Adapted from D. Culler and Yelick's talks

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

Entropy in Key Values

Entropy = \( \sum_i p_i \log p_i \)

\( p_i \) = Probability of key \( i \)

Adapted from D. Culler and Yelick's talks
Local Computation Parameters - Empirical

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

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

Adapted from D. Culler and Yelick’s talks
Odd-Even Merge - classic parallel sort

N values to be sorted
Treat as two lists of M = N/2
Sort each separately
A_0, A_1, A_2, A_3, A_{M-1}
B_0, B_1, B_2, B_3, B_{M-1}

Redistribute into even and odd sublists
A_0, A_2, ..., A_{M-2}, B_0, B_2, ..., B_{M-2}
A_1, A_3, ..., A_{M-1}, B_1, B_3, ..., B_{M-1}

Merge into two sorted lists
E_0, E_1, E_2, E_3, E_{M-1}
O_0, O_1, O_2, O_3, O_{M-1}

Pairwise swaps of E_i and O_i will put it in order

Where’s the Parallelism?

1xN
2xN/2
4xN/4
2xN/2

E_0, E_1, E_2, E_3, E_{M-1}
O_0, O_1, O_2, O_3, O_{M-1}
1xN

Adapted from D. Culler and Yelick’s talks
Mapping to a Butterfly (or Hypercube)

A0 A1 A2 A3 B0 B1 B2 B3
A0 A1 A2 A3 B0 B1 B2 B3
A0 A1 A2 A3 B0 B1 B2 B3
A0 A1 A2 A3
B0 B1 B2 B3
B0 B1 B2 B3
B0 B1 B2 B3
A0 A1 A2 A3
B0 B1 B2 B3
Reverse Order of one list via cross edges
Two sorted sublists
Pairwise swaps on way back

Bitonic Sort

- A bitonic sequence is one that is:
  1. Monotonically increasing and then monotonically decreasing
  2. Or can be circularly shifted to satisfy 1
- A half-cleaner takes a bitonic sequence and produces
  1. First half is smaller than smallest element in 2nd
  2. Both halves are bitonic

Adapted from D. Culler and Yellick's talks
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

```c
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));
```

Adapted from D. Culler and Yelick's talks

---

Bitonic Sort

Block Layout

`lg N/p` stages are local sort
remaining stages involve Block-to-cyclic, local merges
Block-to-cyclic, local merges

Adapted from D. Culler and Yelick's talks

`lg N/p` cols
**Bitonic Sort: time per key**

Adapted from D. Culler and Yelick's talks

**Bitonic: Breakdown**

P= 512, random

Adapted from D. Culler and Yelick's talks
**Bitonic: Effect of Key Distributions**

![Bar chart showing effect of key distributions](image)

- Swap
- Merge Sort
- Local Sort
- Remap C-B
- Remap B-C

$P = 64, N/P = 1 \ M$

Adapted from D. Culler and Yelick's talks

**Column Sort**

1. Sort
2. Transpose - block to cyclic
3. Sort
4. Transpose - cyclic to block w/o scatter
5. Sort
6. Shift
7. Merge
8. Unshift

Treat data like $n \times P$ array, with $n \geq P^2$, i.e. $N \geq P^3$

Adapted from D. Culler and Yelick's talks
Column Sort: Times

Only works for $N \geq P^3$

Column: Breakdown

$P = 64$, random

Adapted from D. Culler and Yelick's talks
**Column: Key distributions**

Adapted from D. Culler and Yelick's talks

- $P = 64$, $N/P = 1M$

**Sequential Radix Sort: Counting Sort**

- Idea: build a histogram of the keys and compute position in answer array for each element

A = [3, 5, 4, 1, 3, 4, 1, 4]

- Make temp array B, and write values into position

  1 1 3 3 4 4 4 5

Adapted from D. Culler and Yelick's talks
### Counting Sort Pseudo Code

- **Counting Sort**

  ```java
  static void countingSort(int[] A) {
      int N = A.length;
      int L = min(A), U = max(A);
      int[] count = new int[U-L+2];
      for (int i = 0; i < N; i += 1) {
      }
      for (int j = 1; j < count.length; j++) {
          count[j] += count[j-1];
      }
      int[] B = new int[N];
      for (int i = 0; i < N; i += 1) {
          B[count[A[i]-L]] = A[i];
          count[A[i]-L] += 1;
      }
      for (int i = 0; i < N; i += 1) {
          A[i] = B[i];
      }
  }
  ```

  **Example:**
  - A = [3, 5, 4, 1, 3, 4, 1, 4]
  - N = 8
  - L = 1, U = 5
  - count = [0, 0, 0, 0, 0, 0]
  - count = [0, 2, 0, 2, 3, 1]
  - count = [0, 2, 2, 4, 7, 8]

  Adapted from D. Culler and Yelick’s talks

---

### Distribution Sort Continued

```java
static void countingSort(int[] A) {
    int[] B = new int[N];
    for (int i = 0; i < N; i += 1) {
        B[count[A[i]-L]] = A[i];
        count[A[i]-L] += 1;
    }
    for (int i = 0; i < N; i += 1) {
        A[i] = B[i];
    }
}
```

**Example:**
- A = [3, 5, 4, 1, 3, 4, 1, 4]
- count = [0, 2, 2, 4, 7, 8]
- B = [0, 0, 0, 0, 0, 0, 0, 0]
- count = [0, 2, 2, 4, 7, 8]
- B = [0, 0, 3, 0, 0, 0, 0, 0]
- count = [0, 2, 3, 4, 8, 8]
- B = [0, 0, 3, 0, 4, 0, 0, 5]
- count = [0, 2, 3, 4, 8, 8]
- B = [1, 0, 3, 0, 4, 0, 0, 5]
- count = [1, 2, 3, 5, 8, 8]
- B = [1, 0, 3, 3, 4, 0, 0, 5]
- count = [1, 2, 4, 5, 8, 8]
- B = [1, 1, 3, 3, 4, 4, 4, 5]

Adapted from D. Culler and Yelick’s talks
Analysis of Counting Sort

• What is the complexity of each step for an n-element array?
  • Find min and max: $\Theta(n)$
  • Fill in count histogram: $\Theta(n)$
  • Compute sums of count: $\Theta(\text{max-min})$
  • Fill in B (run over count): $\Theta(\text{max-min})$
  • Copy B to A: $\Theta(n)$

• So this is a $\Theta(n + m)$ algorithm,
  • where $m=\text{max-min}$
• Great if the range of keys isn’t too large
  • If $m < n$, then $\Theta(n)$ overall

Radix Sort: Separate Key Into Parts

• Divide keys into parts, e.g., by digits (radix)
• Using counting sort on these each radix:
  • Start with least-significant

<table>
<thead>
<tr>
<th>sat</th>
<th>run</th>
<th>sat</th>
<th>pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>saw</td>
<td>pin</td>
<td>saw</td>
<td>run</td>
</tr>
<tr>
<td>tip</td>
<td>tip</td>
<td>pin</td>
<td>sat</td>
</tr>
<tr>
<td>run</td>
<td>sat</td>
<td>pin</td>
<td>saw</td>
</tr>
<tr>
<td>pin</td>
<td>saw</td>
<td>run</td>
<td>tip</td>
</tr>
</tbody>
</table>

Adapted from D. Culler and Yelick's talks
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

Histo-Radix Sort (again)

Local Data

Local Histograms

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

Adapted from D. Culler and Yelick's talks
Radix Sort: Times

Adapted from D. Culler and Yelick's talks

Radix: Breakdown

Adapted from D. Culler and Yelick's talks
**Radix: Key distribution**

Slowdown due to contention in redistribution

Adapted from D. Culler and Yelick's talks

**Radix: Stream Broadcast Problem**

- Processor 0 does only sends
- Others receive then send
- Receives prioritized over sends
  - Processor 0 needs to be delayed

\[ (P-1) \left( 2o + L + (n-1) g \right) \]

Need to slow first processor to pipeline well

Adapted from D. Culler and Yelick's talks
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?

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

Adapted from D. Culler and Yelick’s talks
Sample Sort: Times

Adapted from D. Culler and Yelick's talks

Sample Breakdown

Adapted from D. Culler and Yelick's talks
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

Adapted from D. Culler and Yelick's talks

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

Adapted from D. Culler and Yelick's talks