15-492: Parallel Algorithms **Lecturer:** Guy Blelloch
Topic: Parallel Sorting **Date:** September 18, 2007
Scribe: Charlie Won

7.1 Sample Sort

7.1.1 Steps and Implementation Details

1. Select pivots (to serve as a 'sample') and sort them.
2. Place the keys into buckets delimited by the pivots. If there are n pivots, the keys will be divided into $n + 1$ sets.
3. Sequentially sort the elements within each bucket.
4. Append the results of the sorted buckets.

Advantage: This sorting routine works well on computers with slow communication. Step 2 would be the only step requiring much communication.

Possible Issue: How to select the pivots?

- Randomly - Each bucket would have a size of around $\frac{n}{p}$ on average, but it is quite possible for the largest bucket to be log size larger than another bucket.
- Oversample - Pick $s \cdot p$ candidates, $s > 1$, randomly. Sort the $s \cdot p$ pivots. Select every sth candidate from the sorted result to be a pivot. For any p with s around $log(n)$, it is 'very unlikely' for there to be uneven bucket sizes. The bucket sizes are most likely to range within a constant factor.

7.1.2 Chernoff Bounds

$\mathbf{Pr}[X < \gamma \cdot r \cdot q] \leq e^{-(1-\gamma)^2 \cdot r \cdot q/2}$

- X = random variable representing count of successful trials
- r = # of trials
- q = probability of success
- γ = arbitrary setting $0 \leq \gamma \leq 1$

Theorem: No bucket has over $\frac{\alpha \cdot n}{p}$ elements with high probability.

We would like $\alpha = 2$.

Proof: Let A represent a sorted array. Let this array be divided by $s \cdot p$ randomly selected candidates.

Claim: A bucket size is $\geq \alpha \cdot \frac{n}{p}$ only if there are less than s candidates in this region.

$$\mathbf{Pr}[X < s] = \mathbf{Pr}[Failure]$$

$X = \#$ of pivots in region of size $\alpha \cdot \frac{n}{p}$
$r = \frac{\alpha \cdot n}{p}$ (trials) the number of elements in a particular bucket
$q = \frac{p \cdot s}{n}$ (probability) the likelihood of any individual element being picked
$s = \frac{1}{\alpha} \cdot r \cdot q$

Let X_i be an indicator value of element i being picked, and $X = \sum_{i=1}^{\alpha \cdot \frac{n}{p}} X_i$.

By linearity of expectation, $\mathbf{E}[X] = (s \cdot \frac{p}{n}) \cdot (\alpha \cdot \frac{n}{p}) = \alpha \cdot s$
This is represented by $r \cdot q = \alpha \cdot s$. Therefore,

$$\begin{aligned}
\mathbf{Pr}[X < s] &= \mathbf{Pr}\left[X < \frac{1}{\alpha} \cdot \mathbf{E}[X]\right] \\
&= \mathbf{Pr}\left[X < \frac{1}{\alpha} \cdot r \cdot q\right] \\
&\leq e^{-(1-\frac{1}{\alpha}^2) \cdot r \cdot q/2}
\end{aligned}$$

$\Rightarrow$ if $\alpha = 2$, and letting $s = 4 \cdot \ln(n)$

$$\begin{aligned}
\mathbf{Pr}[X < s] &\leq e^{-1/4 \cdot \alpha s/2} \\
&\leq e^{-1/4 \cdot 2 \cdot 4 \cdot \ln(n)/2} \\
&\leq e^{-\ln(n)} \\
&\leq \frac{1}{n}
\end{aligned}$$

If $s = 8 \ln(n)$, then $\mathbf{Pr}[X < s] < \frac{1}{n^2}$. ■

See p. 21 of handout to see that 64 is the optimal constant.

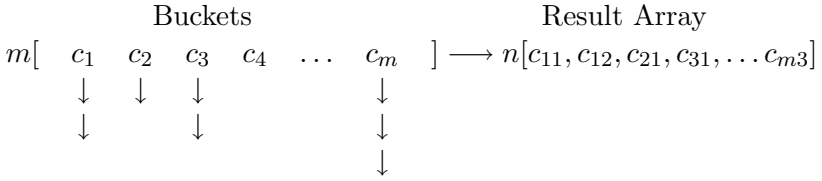
7.1.3 Work and Depth

	Select/Candidate Sort	Distribute	Local Sort	Balance
W	$O(p \cdot \log(n) \cdot \log(p))$	$n \cdot \log(p)$	$n \cdot \log(\frac{n}{p})$	$\frac{n}{p}$
D	$O(p \cdot \log(n) \cdot \log(p))$	$\frac{n}{p} \cdot \log(p)$	$\frac{n}{p} \cdot \log(p)$	$\frac{n}{p}$

7.2 Radix Sort

Radix sort works on integer keys $([1 \dots m]), m < n$.

The elements would be placed into m buckets, and will eventually end up in the result array of size n. If you know the number of elements in each bucket, you know the offsets that the buckets would start at in the result array.



Let $m \leq \frac{n}{p}$.

Steps:

1. Count the number of elements. (plus scan)
2. Define the offsets. (scan on counts)
3. Write to final location.

When counting elements, identify for each processor the area it wants to write into.

If the following table represented the required computations, run plus scan in row major order.

	P_1	P_2	P_3	P_4	$\dots$
0					
1					
2					
3					
$\vdots$					
$m-1$					