15-499: Parallel Algorithms **Lecturer:** Guy Blelloch
Topic: Cilk++ and Sorting **Date:** February 10, 2009
Scribe: Victor Vilchis

9.1 Introduction to Cilk++

Cilk++ adds parallel processing capabilities to C++. Cilk++ keywords can be used by include cilk.h header. Some of Cilk's keywords include:

cilk_spawn Creates parallelism by forking off a parallel task.

cilk_sync There's an implicit one at the end of every function. It acts similar to the join system call of pthread library.

cilk_main: The replacement for the main function of C++. Gives an automatic argument of cilk_set_worker_count to specify the number of threads to be created.

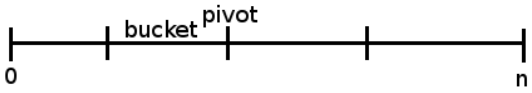
cilk_for: A parallel for that requires that the iteration variable be declared inside the for. Iterators may be used as well.

Cilkscreen detects if threads are reading and writing the same place in memory and generates error messages. To alleviate races resulting from contention for resources, one can use locks. This, however, sequentialized the code. A helpful feature of Cilk++ that cuts down on sequentialization of code due to locking is hyper-objects/reducers. They're a way for the programmer to simultaneously manipulate a shared variable as long as the operator is associative.

9.2 Sorting: Sample sort

Sample sort is asorting algorithm that is good machines with poor communication. It's relatively easy to transform it to run on p workstations. Quicksort is a special case of this algorithm in which $m = 1$. It uses the partitioning paradigm: it uses a set of m pivots sampled from the given set to partition n keys. The algorithm outline is as follows:

1. Select and sort $m < n$ pivots.
2. Associate a bucket with each interval dictated by the pivots.
3. Using binary search, distribute the rest of the keys in to buckets ($m + 1$ buckets)
4. Sequentially sort each bucket.
5. Append results.



This algorithm could be used across several workstations, as step 3 is independent for each bucket. Step 1 would have to be performed serially, and steps 2 and 4 require communication. A possible problem that might impact the performance of this algorithm is having bad pivots. If the pivots do not cut the array in to roughly equally sized pieces, then the processot with the larger chunk ends up being the bottleneck.

If we pick pivots randomly with probability m/n, the average size of the bucket is n/m. However the maximum of the bucket sizes can be $(m/n+1)\log m$ with significant probability. Therefore this choice of pivots can be expected to overload one processor with too much work while others are left idle.

An easy solution to this problem is to oversample: pick around sm keys, when only m are needed ($s > 1$). Sort the sample set and pull out every s-th key to form a smaller pivot set which will then define th bucket interval. If s is large enough (about $O(\log n)$), then as we show below, the work is reasonably balanced.

Theorem 9.2.1 *If the set is oversampled with $s = 12\ln(n)$, then no bucket will be larger than $4n/m$ with probability at least $O(1-1/n^2)$.*

Proof: To show this, think of the array of n elements as a collection of $m/2$ blocks of size $2n/m$ each. The expected number of elements from a block that are selected as pivots is $2s$. Since every s-th pivot is retained for defining buckets, at least one bucket boundary defining pivot is drawn from a given block if the block has at least $s+1$ samples drawn from it.

To bound the probability of the event that less than $s+1$ pivots are drawn from a block, we use Chernoff bounds. Let X_i deonte the probabililty that the i-th element of the block is chosen to be a oversampled pivot. X_i are all independent. Also $E[X] = E[\sum_{i=1}^{2n/m} X_i] = \sum_{i=1}^{2n/m} E[X_i] = 2s$. The probability that $X < (s+1)$ is:

$$\begin{aligned}\mathbf{Pr}[X < (1-1/2)(2s)] &< e^{-(1/2)^2(2s)/2)} \\ &= e^{-(12\ln n)/4} \\ &= 1/n^3\end{aligned} \tag{9.2.1}$$

Since the probability that a given block has no pivot is less than $1/n^3$, the probability that one of the $m/2$ blocks has no pivot is less than $(m/2)(1/n^3) < 1/n^2$ by union bound. Therefore, with probability greater than $1-1/n^2$, each of the $2n/m$ size have blocks have at least one pivot selected to define to bucket boundaries which means that no bukcet will be larger than size $4n/m$.

■