Parallel Bitstream-Based Length Sorting

Robert D. Cameron

May 22, 2011

1 The Idea

Given streams S and E that mark the starts and ends of Names.

We can partition E into a length-sorted set of streams.

First we can partition E into the stream marking ends of Names of length 1, E_1, and the stream of ends of Names of greater than length 1, $E_{>1}$, as follows.

$$\begin{aligned} S_1 &= n(S) \\ E_1 &= S_1 \wedge E \\ E_{>1} &= E \wedge \neg E_1 \end{aligned}$$

Following this, the stream that marks the ends of Names of length 2, E_2, as well as the stream $E_{>2}$ for longer Names can be similarly determined.

$$\begin{aligned} S_2 &= n(S_1) \\ E_2 &= S_2 \wedge E_{>1} \\ E_{>2} &= E_{>1} \wedge \neg E_2 \end{aligned}$$

It may then be desirable to group the Names of length 3 and 4 together. A stream $E_{3,4}$ marking the ends of such names can be determined with a slight variation of the pattern.

$$\begin{aligned} S_{3,4} &= n(n(S_1|S_2)) \\ E_{3,4} &= S_{3,4} \wedge E_{>2} \\ E_{>4} &= E_{>2} \wedge \neg E_{3,4} \end{aligned}$$

Here the stream $S_{3,4}$ is the stream consisting of all positions 3 or 4 positions past a Name start position.

Following this pattern streams for Names of length 5 through 8 are similarly determined.

$$\begin{aligned} S_{5,8} &= n(n(n(S_1|S_2|S_{3,4}))) \\ E_{5,8} &= S_{5,8} \wedge E_{>4} \\ E_{>8} &= E_{>4} \wedge \neg E_{5,8} \end{aligned}$$

2 Follow-up

Once the length sorting above has completed, then we can process names using bitscan techniques.

1. Sequentially scan E_1 to process all names of length 1.
2. Sequentially scan E_2 to process all names of length 2.
3. Sequentially scan $E_{3,4}$ to process all names of length 3 or 4. The actual length may be determined by a bit test of the S stream.
4. Sequentially scan $E_{5,8}$ to process all names of length 5 through 8. The actual length may be determined by a reverse bitscan from each $E_{5,8}$ to the corresponding name start position in S.
5. Sequentially scan $E_{>}8$ to process all names of length greater than 8. This will require both a reverse bit scan to find the name start and a forward scan to find the actual length.

Of course, this scheme shows just one partitioning into length groups; others are possible. Using the log-length strategy, a partition of $E_{>8}$ into $E_{9,16}$ and $E_{>16}$ would probably be worthwhile with XML.

3 Expected Benefits

What is the expected benefit of this technique versus sequentially determining lengths? This avoids allocation of length-based position arrays, the memory operations of insertion of positions into those arrays, bounds-checking on those arrays and maintaining an auxiliary array of lengths of the each array.

A further benefit may be from the programming side: the length sorting can be implemented using our existing Pablo language, avoiding low-level C programming.

4 DIV2 grouping

Given streams S and E that mark the starts and ends of Names. We can partition E into length-sorted groups using div2-length strategy: $G_i = n(E_{i*2-1})|E_{i*2}$

First we calculate T, which marks the two positions after each name. Then we can get G_1 as follows.

$$\begin{aligned} T &= E|n(E) \\ S_2 &= n(n(S)) \\ G_1 &= S_2 \wedge T \end{aligned}$$

$G_i(i > 1)$ can be similarly determined as follows.

$$\begin{aligned} S_{i*2} &= n(n(S_{(i-1)*2} \wedge \neg T)) \\ G_i &= S_{i*2} \wedge T \end{aligned}$$

Once the length sorting above has completed, then we can process names using bitscan techniques.
For each $G_i(i > 0)$:

1. Sequentially scan G_i to find positions $P_i = \{p_{i,0}, p_{i,1}, ..., p_{i,n}, \}$.
2. For each k in $\{1, 2..., n\}$, calculate hash value for name $N[p_{i,k-i*2}]...N[p_{i,k-1}]$

3. Look up hash table. If the name exist, process next name. Otherwise, if $N[p_{i,k-1}]$ is a delimiter (name length is odd), lookup name $N[p_{i,k-i*2}]...N[p_{i,k-2}](i > 1)$ or name $N[p_{1,k-2}]$. If it exists, get its GID g and insert $N[p_{i,k-i*2}]...N[p_{i,k-1}]$ with GID g. If it dosen't exist, insert this name with a new GID and insert $N[p_{i,k-i*2}]...N[p_{i,k-1}]$ with the same GID. If $N[p_{i,k-1}]$ is not a delimiter (name length is even), insert $N[p_{i,k-i*2}]...N[p_{i,k-1}]$ with a new GID.