Weekly Report for Yu Hu's work in week4

February 6, 2005

1 Work1: Analysis of the extension of Weiping Shi's work

The first work of this week is to consider how to expand Weiping Shi's work to handle triple candidates (RAT, Capacitance, Power), i.e. (Q, C, E). In the following description, we use $Q(v, \alpha)$, $C(v, \alpha)$ and $E(v, \alpha)$ to denote the the RAT, downstream capacitance and power dissipation at v under buffer assignment α, respectively.

I tried to organize all candidates in one candidate tree, but failed. The main cause is that there are three possible orders for (Q, C, E) if we organize candidates by non-decreased order of Q as follows (We can find that $Q_1 > Q_2, C_1 < C_2, E_1 < E_2$ won't exist in non-redundant candidate tree, since α_1 dominant α_2.),

1. $Q_1 > Q_2, C_1 > C_2, E_1 > E_2$,
2. $Q_1 > Q_2, C_1 > C_2, E_1 < E_2$,
3. $Q_1 > Q_2, C_1 < C_2, E_1 > E_2$.

To implement the speedup technics in Shi's DAC'03, we must make sure Q and C in the same order. In another words, we can always organize the candidate tree with non-decreased order of both Q and C. Obviously, this is impossible when we add E into consideration.

So, I decide to use a data structure similar as [Lillis_ICCAD'95] to organize all candidates as subsets indexed by E in strictly increased order, such as $\{E_1, T_1\}$, $\{E_2, T_2\}$, $\ldots$, $\{E_n, T_n\}$, where $E_i < E_j$ if $i < j$, and T_i is a candidate tree whose nodes are (Q, C) pairs. Using such a data structure, we can perform two kinds of pruning.

A. **Pruning redundancies in each candidates tree T_i.** For two candidates α_1 and α_2 in the same candidate tree T_i, obviously, $E(\alpha_1) = E(\alpha_2)$, so we can just perform pruning based on (Q,C) pairs. Furthermore, we can organize all (Q,C) pairs by non-decreased order of both Q and C. This makes us able to use Shi's speedup technics in his DAC'03 paper.

B. **Pruning in E list.** We can consider the following 3 operations in buffer insertion respectively:

Merging. New candidates are generated by exploring potential merges so that the new candidates are generated in non-decreased order of E. When each new candidate is generated, its Q and C can be inserted into a range-query tree (a binary search tree, whose nodes are (Q,C), ordered by C) to allow for pruning based on just Q and C. The trick is that by visiting all new candidates in nondecreasing order of E, it is guaranteed that each new candidate added to the range-query tree will be dominated in terms of E by the other candidates already in the tree. Then one only needs to determine additional dominance in Q and C to see whether the candidate should be rejected. This test can be done in time logarithmic in the size of the tree.

Adding a wire. Candidates are updated after a wire is added from the root of the current sub-tree. Similarly as merging, we visit all candidates in nondecreasing order of E, and a temporary range-query tree is maintained. Each time when a candidate is visited, check whether it's redundant in the range-query tree. If not, insert it into the tree.

Adding a buffer. One new candidate is generated in each $\{E_i, T_i\}$ set, which has $max(Q_j - R_b C_j), j \in T_i$. Then we need to check the new candidates' redundancies in non-decreased order of E_i similarly as before.

In both of the two kinds of pruning mentioned above, we can perform **Predictive Pruning** in [Shi_DAC03]. This idea can be expanded as follows to handle power, For candidate α, $P(v, \alpha) = Q(v, \alpha) - R_b C(v, \alpha)$, where

R_b is the resistance of the first buffer predictive used in the upstream of v. Given candidates α_1 and α_2, if $P(v, \alpha_1) > P(v, \alpha_2)$, $C(v, \alpha_1) < C(v, \alpha_2)$, and $E(v, \alpha_1) < E(v, \alpha_2)$, then we say α_1 is b-dominant α_2. The statement is, "If α_1 is b-dominant α_2, then α_2 is redundant and pruned." Furthermore, we have, "If α_1 and $\alpha 2$ do not b-dominant one another, then $P(v, \alpha_1) > P(v, \alpha_2)$ if and only if $Q(v, \alpha_1) > Q(v, \alpha_2)$." So we can use b-dominant to determine the redundancies.

For dual-Vdd buffer insertion, we have $P(v, \alpha) = Q(v, \alpha) - R_b^H C(v, \alpha)$, when the upstream buffer is high-Vdd one, otherwise, we have $P(v, \alpha) = Q(v, \alpha) - R_b^L C(v, \alpha)$. Note that, when a wire is added, we can't decide wether a high-Vdd or low-Vdd buffer will be added in upstream if there're no high-Vdd buffers in downstream, so we need to generate both candidates (In [KingHo_DAC05], it's stated that the high-Vdd buffer peer is always inferior than low-Vdd one. I can't agree with it, since if we calculate the power of the wire by low-Vdd, then the immediate buffer up the wire must be a low-Vdd one, which indicates that the Q after this low-Vdd buffer is smaller than the high-Vdd peer.).

2 Work2: Implement the extension of Weiping Shi's work into BIC package (in processing)

Before implementation of the extension, I read the document and benchmarks of BIC package. After that, I downloaded the GNU-AVL library (http://ftp.gnu.org/gnu/avl/), which contains a well-written package including the implementation of several kinds of binary search trees, such as AVL tree and red-black tree. I will use this package in my future implementation as my basic data structure to organize candidates. I've read the document and part of the source code in the package.

The schedule for the implementation of the extension is as following steps,

1. **Implement the framework of the algorithm.** Firstly, I'll implement the framework of the algorithm, which is based on [Lillis_ICCAD05]. Note that I don't perform any pruning in this step.

2. **Get the candidates distribution in each power value.** As we know, in each subset $\{E_i, T_i\}$, if there exist quite few candidates in T_i, then the efficiency of the speedup technics (Pruning redundancies in each candidates tree T_i.) can be hardly shown. To prove/disprove this worry, I should get a statistic of the candidates number in each power value.

3. **Implement pruning in E list.**

4. **Implement pruning redundancies in each candidates tree T_i.** If there exist many candidates in some T_i in general cases, I will implement pruning redundancies in each candidates tree T_i based on [Shi_DAC03].

At present, I'm processing in step1. The whole implementation is expected to be done in next week.