Physical Design Report in PA3

R08921053 電機研一 梁峻瑋

r08921053@ntu.edu.tw

1.最終的結果表格

	Global HPWL	Global Time	Legal HPWL	Legal Time	Detail HPWL	Detail Time
ibm01-cu85	7.3e8	0s	6.8e8	0s	3.2e8	6s
ibm05	6.5e7	1s	5.9e7	2s	2.9e7	24s

原則上，我參考了 sample code，並且採取 analytical placement—梯度下降法來實作 global placement. 我確實挑選了 wire-length 和 bin-density 的可微分近似函數作為 cost function，並且也計算出他們的 gradient function.

然而，可能是參數調整，或者是計算上有些許問題，導致梯度下降法遲遲無法踏出第一步. 所以，最後我選擇不使用梯度下降法--白白浪費運算時間，而是直接初始化夠好的 module-coordinate，並倚賴 Legalization, Detailed Placement 的部分來完成 Placement.

然而，在這份報告中，我仍然會交代 cost function 選擇與實作的考量，以及最後初始化的方式.

2.實作的程式部分

我把整個程式作業的工作拆成下列的部分來分工運作:

f1 = cost function of wire length

f2 = cost function of all bin's density.

g1[2*i+0] = f1's gradient function to the variable: module[i]'s x-coordinate.

g1[2*i+1] = f1's gradient function to the variable: module[i]'s y-coordinate.

g2[2*i+0] = f2's gradient function to the variable: module[i]'s x-coordinate.

g2[2*i+1] = f2's gradient function to the variable: module[i]'s y-coordinate.

最後，輸出的 f, g=g[i=1:2(numModules)-1]分別為:

$$f = f1+f2.$$

$$g = g1+g2.$$

第一個等式會成立，是基於 cost function 本身就是兩項的總和.

第二個等式會成立，是因為偏微分運算的可加性.

3.**Wire-Length 的函數實作[f1, g1 部分]**

在這裡，我們先給出 f1, g1 函數具體的形式.

Given a small parameter r->0.

f1's x-part:

$$r*\left[\sum_{net[i],i=0}^{i<numNets}\log\left(\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{net[i].pin(j).x(\ \ )}{r}\right)\right)+\sum_{net[i],i=0}^{i<numNets}\log\left(\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{-net[i].pin(j).x(\ \ )}{r}\right)\right)\right]$$

f1's y-part:

$$r*\left[\sum_{net[i],i=0}^{i<numNets}\log\left(\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{net[i].pin(j).y(\ \ )}{r}\right)\right)+\sum_{net[i],i=0}^{i<numNets}\log\left(\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{-net[i].pin(j).y(\ \ )}{r}\right)\right)\right]$$

而 f1 就是"f1's x-part"與"f1's y-part"兩項的總和. 這也就是 HPWL 的近似函數, Log-Sum-Exp Model.

g1[2*k+0]:

$$r*\left[\sum_{net[i],i=0}^{i<numNets}\exp\left(\frac{pin(k).x(\ \ )}{r}\right)\Big/\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{pin(j).x(\ \ )}{r}\right)+\sum_{net[i],i=0}^{i<numNets}\frac{-pin(k).x(\ \ )}{r}\Big/\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{-pin(j).x(\ \ )}{r}\right)\right]$$

g1[2*k+1]:

$$r*\left[\sum_{net[i],i=0}^{i<numNets}\exp\left(\frac{pin(k).y(\ \ )}{r}\right)\Big/\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{pin(j).y(\ \ )}{r}\right)+\sum_{net[i],i=0}^{i<numNets}\frac{-pin(k).y(\ \ )}{r}\Big/\sum_{net[i].pin(j),j=0}^{j<numPins}\exp\left(\frac{-pin(j).y(\ \ )}{r}\right)\right]$$

基本上, ∂/∂a[log(exp(a)+exp(b))] = exp(a)/(exp(a)+exp(b)). 把握這個原則下去進行偏微分，就可以寫出具體的 gradient function 形式了.

顯然地，根據我們描述我們描述的樣子，實作上的演算法可以寫成

```
For i = 0 : numNets-1      //e=net[i]
    For j = 0 : e.numPins-1    //v = e.pin(j)
        Term1 += exp(v.x()/r)
        Term2 += exp(-v.x()/r)
        Term3 += exp(v.y()/r)
        Term4 += exp(-v.y()/r)
    end
    f1+=log(Term1)+log(Term2)+log(Term3)+log(Term4)

    For j = 0:e.numPins-1      //v=e.pin(j)
        id = v.moduleId()
        g1[id*2+0] += exp(v.x()/r)/Term1 – exp(-v.x()/r)/Term2
        g1[id*2+1] += exp(v.y()/r)/Term3 – exp(-v.y()/r)/Term4
    end
f1*=r,      g1[k]*=r for k=0:numNets-1.
```

4.**Bin-Density 的函數實作[f2, g2 部分]**

首先，我們先給出 f2, g2 函數具體的樣子.

$$f2 = \lambda \ * \sum_{\text{bin b=0}}^{255} (\mathrm{Db} - \mathrm{Mb})^2$$

$$\mathrm{Db} = \sum_{\text{module vi}} \frac{F_x(b, vi) * F_y(b, vi)}{\text{bin's area}}$$

$$F_x(b, vi) = \frac{\log\big(\exp\big(a(l-t)\big)+1\big) - \log\left(\exp(-au) + \exp(-at)\right)}{a(1-\exp\left(a*(l-u)\right)}\Bigg|_{t=Cx-\frac{Wv}{2}-Wb}^{t=Cx+\frac{Wv}{2}+Wb}$$

With u = x[2*j+0], l = u+vi.width().

$$F_x(b, vi) = \frac{\log\big(\exp\big(a(l-t)\big)+1\big) - \log\left(\exp(-au) + \exp(-at)\right)}{a(1-\exp\left(a*(l-u)\right)}\Bigg|_{t=Cy-\frac{hv}{2}-hb}^{t=Cy+\frac{hv}{2}+hb}$$

With u = x[2*j+1], l = u+vi.height().

$$g2[2*i+0] = \lambda \ * \sum_{\text{bin b=0}}^{255} (\mathrm{Db} - \mathrm{Mb}) * \frac{\partial}{\partial x[i]}(\mathrm{Db})$$

$$\frac{\partial}{\partial x[i]}(\mathrm{Db}) = \frac{\frac{\partial}{\partial x[i]}[F_x(b, vi)] * F_y(b, vi)}{\text{bin's area}}$$

$$\frac{\partial}{\partial x[i]}[F_x(b, vi)] = \frac{1}{1-\exp\big(a(l-u)\big)}\left[\frac{\exp\big(a(l-t)\big)}{\exp\big(a(l-t)\big)+1} + \frac{\exp\left(-a*u\right)}{\exp(-a*u) + \exp\left(-a*t\right)}\right]_{t=Cx-\frac{Wv}{2}-Wb}^{t=Cx+\frac{Wv}{2}+wb}$$

With u = x[2*j+0], l = u+vi.width().

$$g2[2*i+1] = \lambda \ * \sum_{\text{bin b=0}}^{255} (\mathrm{Db} - \mathrm{Mb}) * \frac{\partial}{\partial y[i]}(\mathrm{Db})$$

$$\frac{\partial}{\partial y[i]}(\mathrm{Db}) = \frac{\frac{\partial}{\partial x[i]}[F_y(b, vi)] * F_x(b, vi)}{\text{bin's area}}$$

$$\frac{\partial}{\partial x[i]}[F_x(b, vi)] = \frac{1}{1-\exp\big(a(l-u)\big)}\left[\frac{\exp\big(a(l-t)\big)}{\exp\big(a(l-t)\big)+1} + \frac{\exp\left(-a*u\right)}{\exp(-a*u) + \exp\left(-a*t\right)}\right]_{t=Cy-\frac{hv}{2}-hb}^{t=Cy+\frac{hv}{2}+hb}$$

With u = x[2*j+1], l = u+vi.height().

實作上，我們把框框切割成 16*16 個 bin，並且編號成 0~255. bin[i]對應到編號(i%16, i/16)，以及左下角坐標(BdyLeft+(i%16)Binwidth, BdyBottom+(i/16)BinHeight)而對於每一個 bin[i], i=0:255，我們可以這樣子計算 f2, g2

```
for(int j=0; j<numM; ++j){
    Module v  = _placement.module(j);
    if((binLeft<x[2*j]+v.width())&&(x[2*j]<binLeft+binWidth)&&(binBott<x[2*j+1]+v.height())&&(x[2*j+1]<binBott+binHeight)){
        double Fx = 0;
        double Fy = 0;

        double t;
        int u = x[2*j], l = u + v.width();
        t = xCenter+v.width()/2+binWidth;
        Fx += (double)1/_a/(1-exp(_a*(l-u)))*( log(exp(_a*(l-t))+1) - log(exp(-1*_a*u)+exp(-1*_a*t)) );

        t = xCenter-v.width()/2-binWidth;
        Fx -= (double)1/_a/(1-exp(_a*(l-u)))*( log(exp(_a*(l-t))+1) - log(exp(-1*_a*u)+exp(-1*_a*t)) );

        u = x[2*j+1], l = u + v.height();
        t = yCenter+v.height()/2+binHeight;
        Fy += (double)1/_a/(1-exp(_a*(l-u)))*( log(exp(_a*(l-t))+1) - log(exp(-1*_a*u)+exp(-1*_a*t)) );

        t = yCenter-v.height()/2-binHeight;
        Fy -= (double)1/_a/(1-exp(_a*(l-u)))*( log(exp(_a*(l-t))+1) - log(exp(-1*_a*u)+exp(-1*_a*t)) );

        Db += (Fx * Fy) / (binWidth * binHeight);

        //Compute dDb[2*j]->g[2*j], dDb[2*j+1]->g[2*j+1] for module[j] in this case
        u = x[2*j], l = u + v.width();
        t = xCenter+v.width()/2+binWidth;
        int S1 = (double)1/(1-exp(_a*(l-u)))*(exp(_a*(l-t))/(1+exp(_a*(l-t))) + exp(-1*_a*u)/(exp(-1*_a*u)+exp(-1*_a*t)));

        t = xCenter-v.width()/2-binWidth;
        int S2 = (double)1/(1-exp(_a*(l-u)))*(exp(_a*(l-t))/(1+exp(_a*(l-t))) + exp(-1*_a*u)/(exp(-1*_a*u)+exp(-1*_a*t)));

        u = x[2*j+1], l = u + v.height();
        t = yCenter+v.height()/2+binHeight;
        int S3 = (double)1/(1-exp(_a*(l-u)))*(exp(_a*(l-t))/(1+exp(_a*(l-t))) + exp(-1*_a*u)/(exp(-1*_a*u)+exp(-1*_a*t)));

        t = yCenter-v.height()/2-binHeight;
        int S4 = (double)1/(1-exp(_a*(l-u)))*(exp(_a*(l-t))/(1+exp(_a*(l-t))) + exp(-1*_a*u)/(exp(-1*_a*u)+exp(-1*_a*t)));

        dDb[2*j] = (S1-S2)*Fy / (binWidth * binHeight);
        dDb[2*j+1] = Fx*(S3-S4) / (binWidth * binHeight);
    }
}
```

5.選擇 Log-Sum-Exp Model 與 Sigmoid Model 的原因與比較

關於 Wire-Length 部分，原則上我只能選擇 HPWL 的近似函數，或者是 Squared-Euclidean-distance．然而考量到 S.E.d.會極大的高估 cost function，而且還需要調整每一條邊之間的比重，判斷實作上較為複雜且誤差高．特別是，在此我們的 module size 高達百萬等級，情況會更嚴重.

而在於 HPWL 的兩種近似模型，我考量的依據是”gradient function”計算的難易度．更直白地說，由於偏微分具有可加性，撇除 sigma 不管，我們只需要處理 log()的微分[Log-Sum-Exp Case]，或者是多項式函數的微分[Weighted-Average Case]．兩者相較之下，顯然 log()函數的微分會好處理非常多，甚至只要觀察出 log(exp(a)+exp(b))的微分規律，就可以寫下 gradient function 的形式．所以，我選擇用 Log-Sum-Exp case 來實作．儘管我們知道, WA 模型可以更快的逼近到極限，而且他的品質也比較好.

關於 Bin-Density 部分，原則上就是 Bell-shaped 與 Sigmoid 兩種模型．我不選擇 Bell-shaped 的原因，在於我們基本上只能勉強用分段函數寫出積分後的形式.

然而，要對分段函數作偏微分，並不是一件容易處理的任務. 至少是需要花費一定的功夫.

此外，我選擇 sigmoid 的原因，在於他的 density function 十分簡單，基本上就是 1/(1+exp(-a(x-l))(1+exp(-a(u-x))). 而且它的積分也可交由 wolfram，或計算器來計算，驗算起來也不太困難. 至於偏微分的部分，也只需要對 log()的組合函數進行偏微分，幾乎跟 Log-Sum-Exp 的情況是一樣的. 但相對的，我就需要面臨，在中央附近，函數曲線過於平滑，導致 cost function 無法脫離的問題.

6.改進的想法與點子

就我在本次作業，最後我採取初始化的方法來排列 module. 也就是，我給所有的 module，個別一個在範圍內 random 出來的坐標，並且直接讓做 Legalization 等部分.

我認為, analytical placement 失敗的原因，應該在於 cost function 計算上有一些過失，有一些問題. 例如，在隨機的初始化後，本來預期能成功踏出梯度下降法的第一步，然而卻得出-nan 的結果，合理推測應當是參數設定不佳，或者是計算本身就有問題. 另一方面, sample code 中的 solver 函數，或許可以調整成，從所有的 gradient 裡面，從梯度最大的開始，每個方向都嘗試一次，直到 cost function 下降為止. 而不是最大的梯度失敗後，就直接停止. 這樣的容錯率會比較高.