- Vc master branch, build with icc
- Three implementations

1. Simple, automatic vectorization

```
double z_ = 0;
for (int i = 0; i < nrows; i++) {
    double r2 = pow(x - mXArray[i], 2) + pow(y - mYArray[i], 2);
    z_ += mWArray[i] * ur(r2);
}
return z_;
```

2. Vc, explicit vectorization

```
Vc::double_v z_v_(Vc::Zero);
for (int i = 0; i < mXArray_v.vectorsCount(); i++) {
    Vc::double_v mX_v = mXArray_v.vector(i);
    Vc::double_v mY_v = mYArray_v.vector(i);
    Vc::double_v r2_v = (x_v - mX_v) * (x_v - mX_v) + (y_v - mY_v) * (y_v - mY_v);
    z_v_(r2_v > 0) += mWArray_v.vector(i) * r2_v * Vc::log(r2_v);
}
return z_v_.sum();
```

3. std::valarray, dependent on compiler implementation

```
x_val = x;
y_val = y;
z_val = 0.0;
valarray<double> r2_val = (x_val - mXArray_val) * (x_val - mXArray_val) +
    (y_val - mYArray_val) * (y_val - mYArray_val);
z_val[r2_val > 0.0] = mWArray_val * r2_val * log(r2_val);
return z_val.sum();
```

- O3 optimization

gcc 4.8.2

```
nElements = 2000, nVectors  = 1000
Vc class uses    1.544626s
autovec  uses    5.648405s
valarray uses    5.676962s
```

icc 15.0.4

```
nElements = 2000, nVectors  = 1000
Vc class uses    6.913944s
autovec  uses    0.565094s
valarray uses    2.179076s
```