Final Project

Compiler Optimization Differences

```
for (int j = l; j <= r; j++) {
    int x = i - j, y = j;
    dp[2][j + 1] = dp[0][j] + (A[x] == B[y] ? match : miss);
    dp[2][j + 1] = max(
        dp[2][j + 1],
        (dp[1][j] > dp[1][j + 1] ? dp[1][j] : dp[1][j + 1]) + shift);
}
```

The code above will result in different behaviors with the following declarations:

1. C-like array, 10s:

```
DP_TYPE dp[3][int((na + nb) * 1.2];
```

2. Pointers, 2m:

```
DP_TYPE ** dp = (DP_TYPE **)malloc(3 * sizeof(DP_TYPE *));
for (int i = 0; i < 3; ++i)
    dp[i] = (DP_TYPE *)malloc(int((na + nb) * 1.2) * sizeof(DP_TYPE));
```

3. `std::vector`, 2m:

```
vector<vector<DP_TYPE>> dp(3, vector<DP_TYPE>(int((a.length() + b.length()) * 1.2),
0));
```

Potential Problems

1. Stack versus heap
    1. Reason: Accessing stack is much faster than heap.
    2. Test: Move `DP_TYPE dp[3][int((na + nb) * 1.2];` from function scope to global scope.
    3. Result: Nothing changed.
2. Different compiler optimizations
    1. Reason: Compiler optimizations has a strong impact on a programs performance.
    2. Test: Compare optimizations applied to different declarations with `-fopt-info-optall`.
    3. Result: Loop vectorization was only applied to `DP_TYPE dp[3][int((na + nb) * 1.2];`

Solution

Google / StackOverflow indicated that **complicated** loops will not be vectorized by compilers, and `[]` is a complex operator to `std::vector` while iterators are not.
Changing the loop to the following solves the problem:

```
auto it_2 = dp[2].begin() + j + 1, it_2end = dp[2].begin() + r + 2;
auto it_1 = dp[1].begin() + j, it_11 = dp[1].begin() + j + 1, it_0 = dp[0].begin() +
j;
auto it_a = a.begin() + i - j, it_b = b.begin() + j;
for (;it_2 != it_2end; ++it_2)
```

```
{
    *it_2 = *it_0 + (*it_a == *it_b ? match : miss);
    auto it_shift = (*it_1 > *it_11 ? *it_1 : *it_11) + shift;
    *it_2 = (it_shift > *it_2 ? it_shift : *it_2);
    ++j, ++it_1, ++it_11, ++it_0, --it_a, ++it_b;
}
```

Inconsistent Results between Different Runs

Reason: Race condition
Test: Remove `static` decorations inside `bio_match` function
Result: Results became consistent between runs and the performance improved.

DP Calculation Too Slow

Solution: Restrict the size of dynamic programming calculations with bandwidth.

Bandwidth	0.15	0.03	0.003
Speed	1x	5x	40x
Correctness	1	2	3

Bandwidth restriction down side: minimum score may be compromised.

Potential Improvements

GPU acceleration

Unfortunately, ROCm hasn't support Vega M GPUs.