Drought Risk idex from Humidity Analysis

DRI Computation

- **Inputs**
  - Time series ความชื้นสัมพัทธ์ต่อเมือง (0–100)
  - จำนวนจุดข้อมูล N
  - เกณฑ์ “แห้ง” = 30%

- **Dry-Day Frequency (HFI)**
  - HFI = count(h[t] < 30) / N

- **Statistical Measures (ใช้สร้างดัชนีย่อย)**
  - Mean humidity:
    h_bar = (1/N) * Σ_{t=1..N} h[t]
  - HSI (Severity):
    HSI = 0.6 * (1 - h_bar/100) → clip [0,1]
  - HTI (Trend):
    แบ่งซีรีส์เป็น 2 ครึ่ง: h1_bar, h2_bar
    slope = ((h2_bar - h1_bar) / (N/2)) * 100
    HTI = max(0, -slope/2) → clip [0,1]
  - HVI (Volatility):
    Vol = (1/(N-1)) * Σ_{t=2..N} |h[t] - h[t-1]|
    HVI = Vol / 100 → clip [0,1]

- **Final Index (DRI)**
  - DRI = 0.35*HFI + 0.30*HSI + 0.20*HTI + 0.15*HVI → clip [0,1]

Sequential Algorithm

- **Data Loading — O(R*C)**
  - Read & parse CSV: R = time points (rows), C = cities (columns)

- **Statistical Measures — O(R*C)**
  - Mean humidity (h_bar) = (1/N) * Σ_{t=1..N} h[t]
  - Dry-day count (HFI) = count(h[t] < 30)
  - Volatility (HVI) = (1/(R-1)) * Σ_{t=2..R} |h[t] - h[t-1]|
  - Trend (HTI) = avg(half2) - avg(half1) → simple slope

- **DRI Aggregation — O(C)**
  - DRI = 0.35*HFI + 0.30*HSI + 0.20*HTI + 0.15*HVI (clip [0,1])

- **Ranking (optional) — O(C log C)**
  - Sort cities by DRI (or O(C log k) for Top-k)

- **Overall**
  - Time: O(R*C) (+ O(C log C) if sorting)
  - Space: Θ(R*C) if load-all, or Θ(R) per city with streaming

Parallel Algorithm (CUDA)

- **Data Loading (H2D / D2H) — O(R*C)**
  - Copy humidity matrix Host↔Device (time ~ bytes / PCIe bandwidth; do once)

- **Statistical Measures (Accumulate on GPU) — Work = O(R*C)**
  - Parallel time ≈ O((R*C)/(kB*T)) + O(log T)
    * city → blocks (kB per city), time → threads (T per block)
    * coalesced reads + shared/warp reduction (ลด atomics)
  - Metrics computed:
    Mean (h_bar), HFI, HVI = (1/(R-1)) * Σ|Δh|, HTI = simple slope (half2–half1)

- **DRI Aggregation — O(C)**
  - Parallel time ≈ O(C / T_f) + O(log T_f)
  - DRI = 0.35*HFI + 0.30*HSI + 0.20*HTI + 0.15*HVI (clip [0,1])

- **Ranking (optional) — O(C log C)**
  - GPU sort or copy back to CPU and sort

- **Overall (vs Sequential)**
  - T_GPU ≈ T_H2D + (R*C)/(kB*T*η) + (C/T_f) + T_D2H
  - When R*C is large and config T≈256, kB≥4 with good coalescing/reduction ⇒
    Speedup S = T_CPU / T_GPU > 1 (ส่วนคำนวณหลักถูกหารด้วย kB*T)

- **Space**
  - Device buffers ~ O(R*C) (or streaming-by-city to reduce memory)

Sequential Algorithm

คำนวณ HFI/HSI/HTI/HVI → DRI

```
// Calculate DRI for a single city
void calculateDRIForCity(CityDroughtRisk* city, float** humidity_data, int* record_count) {
    if (city->total_records == 0) return;

    // Calculate basic statistics
    double sum = 0.0;
    long dry_days = 0;

    for (int i = 0; i < city->total_records; i++) {
        float humidity = humidity_data[city->data_column][i];
        sum += humidity;
        if (humidity < 30.0f) {
            dry_days++;
        }
    }

    city->avg_humidity = sum / city->total_records;
    city->dry_days = dry_days;

    // Calculate HFI - Humidity Frequency Index
    double HFI = (double)dry_days / city->total_records;

    // Calculate HSI - Humidity Severity Index
    double avg_ratio = city->avg_humidity / 100.0;
    double HSI = (1.0 - avg_ratio) * 0.6;  // Simplified version

    // Calculate HTI - Humidity Trend Index (simplified linear trend)
    if (city->total_records > 100) {
        int first_half = city->total_records / 2;
        double first_avg = 0.0, second_avg = 0.0;

        for (int i = 0; i < first_half; i++) {
            first_avg += humidity_data[city->data_column][i];
        }
        first_avg /= first_half;

        for (int i = first_half; i < city->total_records; i++) {
```

Parallel Algorithm

/

```cpp
// ========== Phase 1: Accumulate (multi-block per city) ==========
__global__ void accumulate_city_stats(
    const float* __restrict__ humidity, // [num_cities * stride]
    const int*   __restrict__ counts,   // [num_cities]
    double* __restrict__ sum_out,       // [num_cities]
    double* __restrict__ absdiff_out,   // [num_cities]
    double* __restrict__ first_out,     // [num_cities]
    double* __restrict__ second_out,    // [num_cities]
    long long* __restrict__ dry_out,    // [num_cities]
    int stride, int num_cities, int kBlocksPerCity
){
    int global_block = blockIdx.x;
    int city = global_block % num_cities;
    int n = counts[city];
    if (n <= 0) return;

    const float* base = humidity + city * stride;
    int tid = threadIdx.x;
    int block_id_for_city = global_block / num_cities;
    int cityWideStride = blockDim.x * kBlocksPerCity;

    double sum=0.0, absdiff=0.0, first=0.0, second=0.0;
    long long dry=0;

    int half = n/2;
    bool do_trend = (n > 100);

    for (int i = tid + block_id_for_city * blockDim.x; i < n; i += cityWideStride) {
        float h = __ldg(&base[i]);
        sum += h;
        if (h < 30.0f) dry++;
        if (i > 0) {
            float prev = __ldg(&base[i - 1]);
            absdiff += fabsf(h - prev);
        }
        if (do_trend) {
            if (i < half) first += h;
            else          second += h;
        }
    }
```

```cpp
    // warp reduce
    sum     = warpReduceSumD(sum);
    absdiff = warpReduceSumD(absdiff);
    first   = warpReduceSumD(first);
    second  = warpReduceSumD(second);
    dry     = warpReduceSumLL(dry);

    __shared__ double s_sum[32], s_abs[32], s_fst[32], s_snd[32];
    __shared__ long long s_dry[32];

    if ((tid & 31) == 0) {
        int wid = tid >> 5;
        s_sum[wid] = sum;
        s_abs[wid] = absdiff;
        s_fst[wid] = first;
        s_snd[wid] = second;
        s_dry[wid] = dry;
    }
    __syncthreads();

    if (tid < 32) {
        int limit = blockDim.x >> 5;
        double bsum=0.0, babs=0.0, bfst=0.0, bsnd=0.0;
        long long bdry=0;
        #pragma unroll
        for (int w=0; w<limit; ++w) {
            bsum += s_sum[w];
            babs += s_abs[w];
            bfst += s_fst[w];
            bsnd += s_snd[w];
            bdry += s_dry[w];
        }
        if (tid == 0) {
            ATOMIC_ADD_D(&sum_out[city],     bsum);
            ATOMIC_ADD_D(&absdiff_out[city], babs);
            ATOMIC_ADD_D(&first_out[city],   bfst);
            ATOMIC_ADD_D(&second_out[city],  bsnd);
            ATOMIC_ADD_LL(&dry_out[city],    bdry);  // signed -> unsigned cast
        }
    }
}
```

Parallel Algorithm

```
__global__ void finalize_city_stats(
    const double* __restrict__ sum_in,
    const double* __restrict__ absdiff_in,
    const double* __restrict__ first_in,
    const double* __restrict__ second_in,
    const long long* __restrict__ dry_in,
    const int* __restrict__ counts,
    double* __restrict__ avg_out,
    double* __restrict__ slope_out,
    double* __restrict__ vol_out,
    double* __restrict__ dri_out,
    int num_cities
){
    int city = blockIdx.x * blockDim.x + threadIdx.x;
    if (city >= num_cities) return;

    int n = counts[city];
    if (n <= 0) {
        avg_out[city]=0.0; slope_out[city]=0.0; vol_out[city]=0.0; dri_out[city]=0.0;
        return;
    }

    double avg = sum_in[city] / (double)n;
    double vol = (n > 1) ? (absdiff_in[city] / (double)(n - 1)) : 0.0;

    int half = n/2;
    double slope = 0.0;
    if (n > 100 && half > 0 && (n - half) > 0) {
        double first_avg  = first_in[city]  / (double)half;
        double second_avg = second_in[city] / (double)(n - half);
        slope = (second_avg - first_avg) / (double)half * 100.0;
    }

    double HFI = (double)dry_in[city] / (double)n;
    double HSI = (1.0 - (avg / 100.0)) * 0.6;
    double HTI = (slope < 0.0) ? (-slope / 2.0) : 0.0;
    double HVI = vol / 100.0;

    if (HSI < 0.0) HSI = 0.0; if (HSI > 1.0) HSI = 1.0;
    if (HTI < 0.0) HTI = 0.0; if (HTI > 1.0) HTI = 1.0;
    if (HVI < 0.0) HVI = 0.0; if (HVI > 1.0) HVI = 1.0;

    double DRI = 0.35*HFI + 0.30*HSI + 0.20*HTI + 0.15*HVI;
    if (DRI < 0.0) DRI = 0.0; if (DRI > 1.0) DRI = 1.0;

    avg_out[city]   = avg;
    slope_out[city] = slope;
    vol_out[city]   = vol;
    dri_out[city]   = DRI;
}
```