SEQUENTIAL Parallel Lab

```
void work_it_seq(long *old, long *new) {
  int i, j, k;
  int u, v, w;
  long compute_it;
  long aggregate=1.0;

  for (i=1; i<DIM-1; i++) {
    for (j=1; j<DIM-1; j++) {
      for (k=1; k<DIM-1; k++) {
        compute_it = old[i*DIM*DIM+j*DIM+k] * we_need_the_func();
        aggregate+= compute_it / gimmie_the_func();
      }
    }
  }

  printf("AGGR:%ld\n",aggregate);

  for (i=1; i<DIM-1; i++) {
    for (j=1; j<DIM-1; j++) {
      for (k=1; k<DIM-1; k++) {
        new[i*DIM*DIM+j*DIM+k]=0;
        for (u=-1; u<=1; u++) {
          for (v=-1; v<=1; v++) {
            for (w=-1; w<=1; w++) {

new[i*DIM*DIM+j*DIM+k]+=old[(i+u)*DIM*DIM+(j+v)*DIM+(k+w)];
            }
          }
        }
        new[i*DIM*DIM+j*DIM+k]/=27;
      }
  }}

  for (i=1; i<DIM-1; i++) {
    for (j=1; j<DIM-1; j++) {
      for (k=1; k<DIM-1; k++) {
        u=(new[i*DIM*DIM+j*DIM+k]/100);
        if (u<=0) u=0;
        if (u>=9) u=9;
        histogrammy[u]++;
      }
    }
  }
}
```

FAST Parallel Lab

```
void work_it_par(long *old, long *new) {
    const int DIM_SQUARED = DIM * DIM;
    const int TILE_SIZE = 4;
    int memory_address, temp_memory_address, i, j, k, ii, jj, kk = 0;
    long index0, index1, index2, index3, index4, index5 ,index6,
index7, index8, index9, u, temp_sum, compute_it = 0;
    long aggregate = 1.0;

    #pragma omp parallel for private(j, k, ii, jj, kk, u, compute_it,
memory_address, temp_memory_address) reduction(+: aggregate)
reduction(+:temp_sum) reduction(+:index0) reduction(+:index1)
reduction(+:index2) reduction(+:index3) reduction(+:index4)
reduction(+:index5) reduction(+:index6) reduction(+:index7)
reduction(+:index8) reduction(+:index9)
    for (i=1; i<DIM-1; i+= TILE_SIZE) {
    for (j=1; j<DIM-1; j+= TILE_SIZE) {
    for (k=1; k<DIM-1; k+= TILE_SIZE) {
        for(ii = i; (ii < i + TILE_SIZE && ii < DIM - 1); ii ++) {
        for(jj = j; (jj < j + TILE_SIZE && jj < DIM - 1); jj ++) {
        for(kk = k; (kk < k + TILE_SIZE && kk < DIM - 1); kk ++) {

            memory_address = ii * DIM_SQUARED + jj * DIM + kk;
            compute_it = old[memory_address] * we_need_the_func();
            aggregate += compute_it / gimmie_the_func();
            temp_sum = 0;

            temp_memory_address = memory_address - DIM_SQUARED;
            temp_memory_address -= DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];
            temp_memory_address += DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];
            temp_memory_address += DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];

            temp_memory_address = memory_address;
            temp_memory_address -= DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];
            temp_memory_address += DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
```

```
                temp_sum += old[temp_memory_address+1];
            temp_memory_address += DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];

            temp_memory_address = memory_address + DIM_SQUARED;
            temp_memory_address -= DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];
            temp_memory_address += DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];
            temp_memory_address += DIM;
                temp_sum += old[temp_memory_address-1];
                temp_sum += old[temp_memory_address];
                temp_sum += old[temp_memory_address+1];

            temp_sum /= 27;
            new[memory_address] = temp_sum;
            u = temp_sum / 100;

            if     (u <= 0) index0++;
            else if(u == 1) index1++;
            else if(u == 2) index2++;
            else if(u == 3) index3++;
            else if(u == 4) index4++;
            else if(u == 5) index5++;
            else if(u == 6) index6++;
            else if(u == 7) index7++;
            else if(u == 8) index8++;
            else if(u >= 9) index9++;
        }}}
    }}}
    printf("AGGR:%ld\n",aggregate);

    histogrammy[0] = index0;
    histogrammy[1] = index1;
    histogrammy[2] = index2;
    histogrammy[3] = index3;
    histogrammy[4] = index4;
    histogrammy[5] = index5;
    histogrammy[6] = index6;
    histogrammy[7] = index7;
    histogrammy[8] = index8;
    histogrammy[9] = index9;

}
```