Assignment 9 – Task 1

- OpenMP-parallel 2D Jacobi smoother:

```
#pragma omp parallel for private(j)
for(i=1; i<N-1; ++i)
  for(j=1; j<N-1; ++j)
    T[t1][i][j] = 0.25*(T[t0][i-1][j]+T[t0][i+1][j]
                        +T[t0][i][j-1]+T[t0][i][j+1]);
```

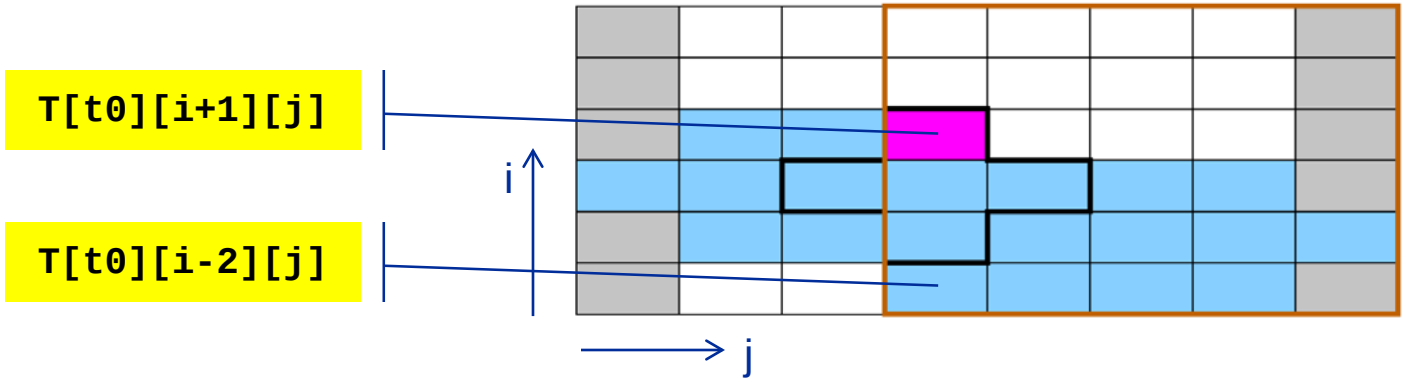
- Expected maximum performance on one Emmy socket at 24 B/LUP:

  $P = b_s/B_c$ = 41 Gbyte/s / (24 byte / 4 Flops) = **6.83 GFlop/s**

  or equivalently, **1.71 GLUP/s**

Assignment 9 – Task 1

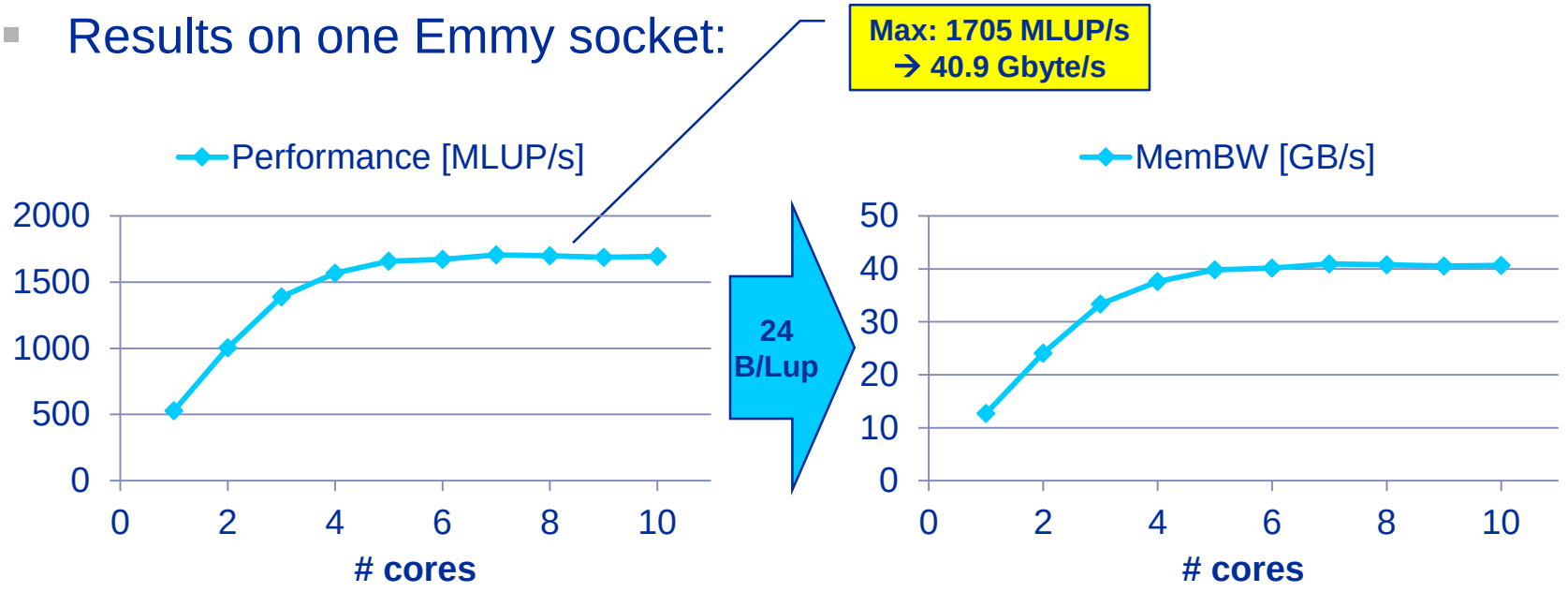
- Layer condition @ 4000x4000 ?



- 3 layers = 3 x 4000 x 8 Bytes = 96 KiB
  → layer condition fulfilled for L2 and L3 cache (but not in L1)
- → Spatial blocking will not result in better *saturated* performance @ 4000x4000

Assignment 9 – Task 1

- Results on one Emmy socket:



- Typical behavior for bandwidth-limited code!
- Roofline model is accurate in the saturated regime
- No scaling across sockets if only the main loop is parallel!
  - Fix by paralell first-touch initialization

Assignment 9 – Task 2: Jacobi 3D

Correct code:

```
#pragma omp parallel private(iter)
{
  for(iter=0; iter<maxiter; ++iter) {
    #pragma omp for schedule(runtime) private(j,i)
    for(k=1; k<N-1; ++k) {
      for(j=1; j<N-1; ++j) {
        for(i=1; i<N-1; ++i) {
          f[t1][k][j][i] = 1./6. * (f[t0][k-1][j][i]+ f[t0][k+1][j][i]+
                                    f[t0][k][j-1][i]+ f[t0][k][j+1][i]+
                                    f[t0][k][j][i-1]+ f[t0][k][j][i+1]);
        }
      }
    }
    #pragma omp single
      swap(t0,t1);
  }
}
```

Assignment 9 – Task 2: Jacobi 3D

- Performance model for Ivy Bridge (Emmy) socket
  - b_S = 41 GB/s
  - CS = 25 MB (L3 cache), 10 cores
  - N = 350 (2 x 350^3 x 8 B = 686 MB → out of cache)
  - **`OMP_SCHEDULE = static`**
- Layer condition:

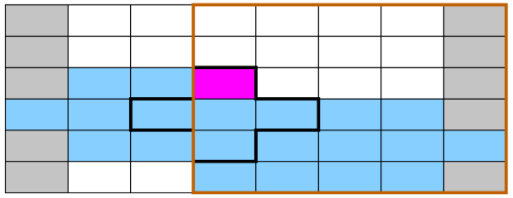
  10 (threads) x 350^2 x 3 (layers) x 8 (bytes) = 29.4 MB

  This is larger than CS/2, so the layer condition is not fulfilled in L3 → B_c = 40 Bytes/LUP

  This holds also for the L3-L2 and the L2-L1 data paths → Memory BW is the relevant bottleneck

- Performance: $P = b_S/B_c$ = 1025 MLUP/s.

Assignment 9 – Task 2: Jacobi 3D



- N=200 layer condition:

  10 (threads) x 200^2 x 3 (layers) x 8 (bytes) = 9.6 MB
  → Layer condition fulfilled in L3, B_c = 24 Bytes/LUP

  → P = b_S/B_c = 1708 MLUP/s.

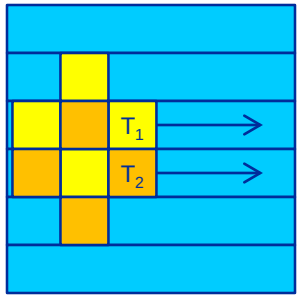
- `f[k][j][i][t]` data layout?

  → read and write streams both have a stride of two, but are interleaved
  → But: all cache lines are modified! RHS data will be evicted along with LHS data
  → B_c = 32 Bytes/LUP even if layer condition is OK!

Assignment 9 – Task 2: Jacobi 3D

- N = 350, **`OMP_SCHEDULE = static,1`**
  - Shared L3 cache can hold data needed by neighboring threads –

    → New (relaxed) layer condition:
    (10+2) x 350² x 8 (bytes) = 11.8 MB
    → B_c = 24 Bytes/LUP



- Inner loop parallelization with barrier cost of 2000 cycles
  - Layer condition for 1 thread holds because of inner loop parallelization!
  - N = 350 → P = 1708 MLUP/s → 585 ps/LUP → 1.3 cy/LUP
  - One line update (348 LUPs) costs 452 cy without the barrier, but 2452 cy with the barrier
    → P = (348 LUPs / 2452 cy) * 2.2 Gcy/s = **312 MLUP/s**

Assignment 9 – Task 2: Jacobi 3D

- Can we beat the 1025 MLUP/s with inner loop parallelization?

  Work (# LUPs) = *W*:

$$\left[\frac{W}{\left[2000\ \mathrm{cy} + 2.2\ \mathrm{Gcy/s} \cdot \frac{W}{1.708\ \mathrm{GLUP/s}}\right]}\right] \cdot 2.2\,\frac{\mathrm{Gcy}}{\mathrm{s}} = 1.025\ \mathrm{GLUP/s}$$

  Solve for *W* → ***W* = 2330 LUPs**

  But: layer condition in danger! 2330*350*3*8 B = 19.6 MB **> CS/2**
  → simply making the inner dimension larger will not help
  → solution?

- nowait clause can eliminate barrier, but need to sync after full sweep!

Assignment 9 – Task 2: Jacobi 3D

- Using nowait

```
#pragma omp parallel private(iter,k,j)
{
  for(iter=0; iter<maxiter; ++iter) {
    for(k=1; k<N-1; ++k) {
      for(j=1; j<N-1; ++j) {
        #pragma omp for schedule(static) nowait
        for(i=1; i<N-1; ++i) {
          f[t1][k][j][i] = 1./6. * (f[t0][k-1][j][i]+ f[t0][k+1][j][i]+
                                    f[t0][k][j-1][i]+ f[t0][k][j+1][i]+
                                    f[t0][k][j][i-1]+ f[t0][k][j][i+1]);
        }
      }
    }
    #pragma omp barrier
    #pragma omp single
      swap(t0,t1);
  }
}
```