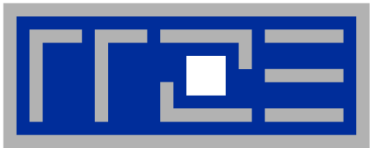
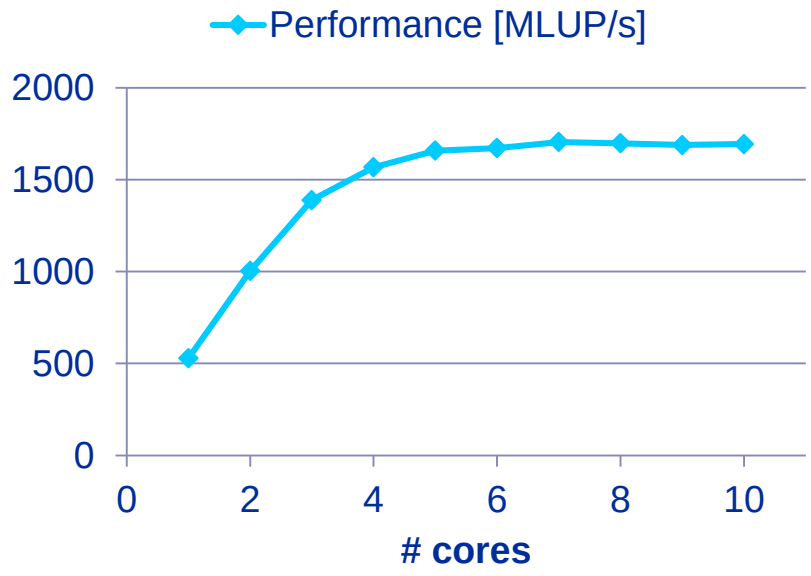
Assignment 6 – Task 1



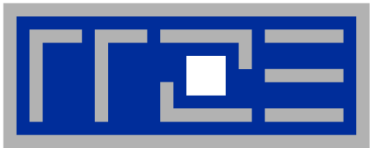
- OpenMP-parallel 2D Jacobi smoother:

```
#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]);
```



Typical behavior for bandwidth-limited code!

Assignment 6 – Task 1



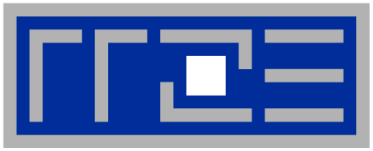
- OpenMP-parallel 2D Jacobi smoother:

```
#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]);
```

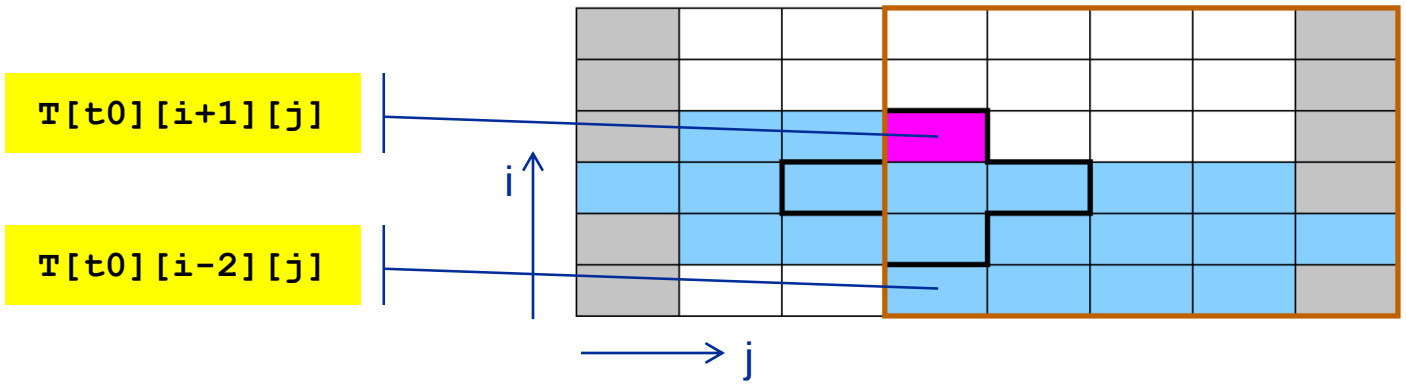
- Expected maximum performance on one Emmy socket at 24 B/LUP:

$P^*B_c = b_s$ = 1705 MLUP/s * (24 B/LUP) = **40.92 GB/s**

Assignment 6 – Task 1



- Layer condition @ 4000x4000 ?

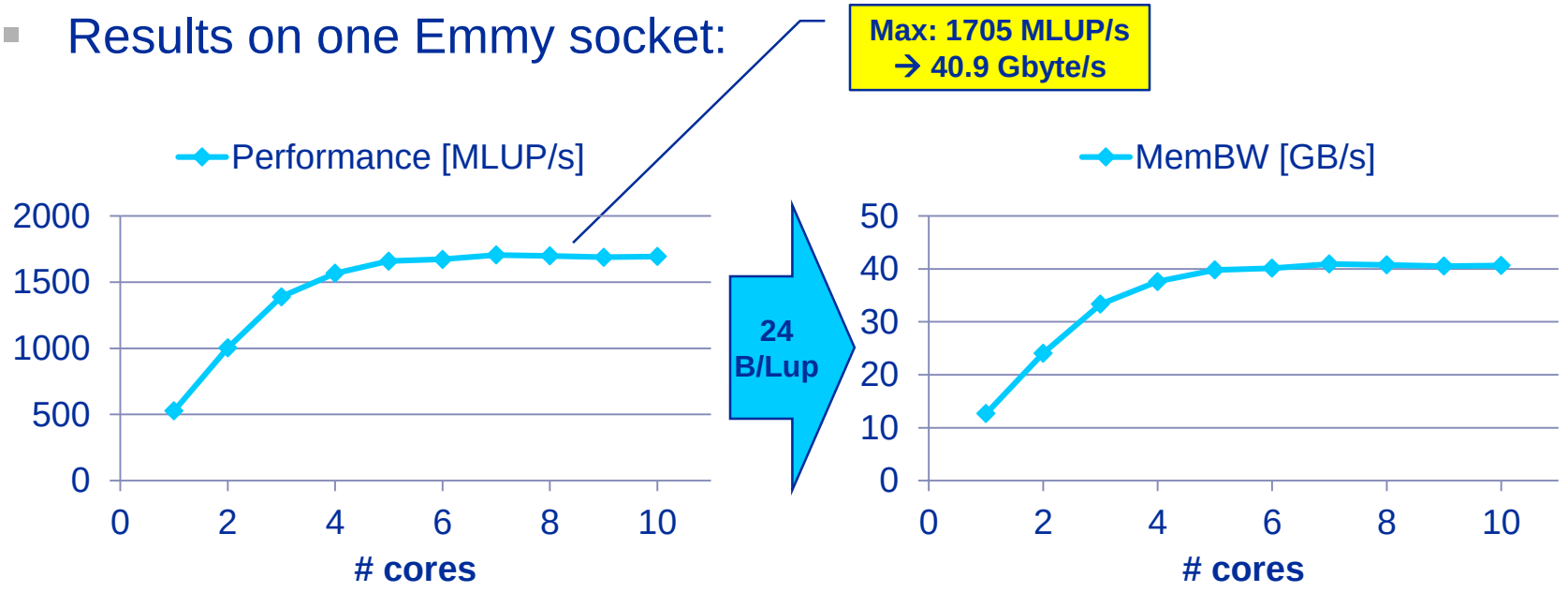


- 3 layers = 3 x 4000 x 8 Bytes = 96 KiB
  → layer condition fulfilled for L2 and L3 cache (but not in L1)
  L1: 40B/LUP; L2: 32B/LUP; L3&MEM: 24B/LUP

- → Spatial blocking will not result in better *saturated* performance @ 4000x4000

Assignment 6 – Task 1

- Results on one Emmy socket:

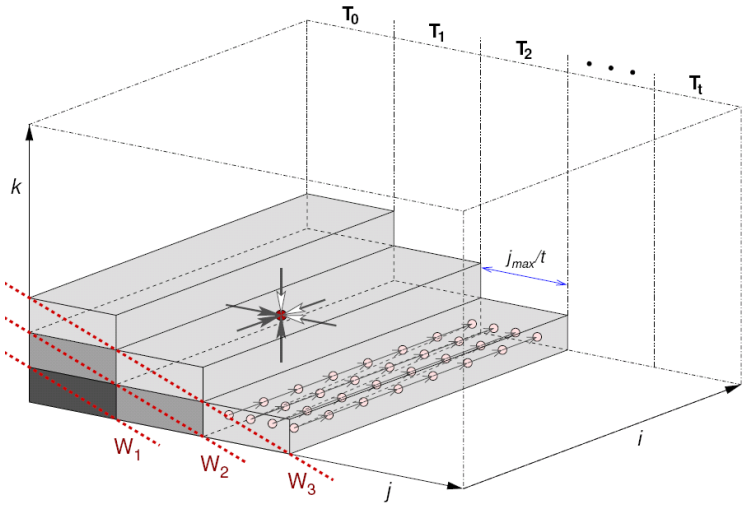


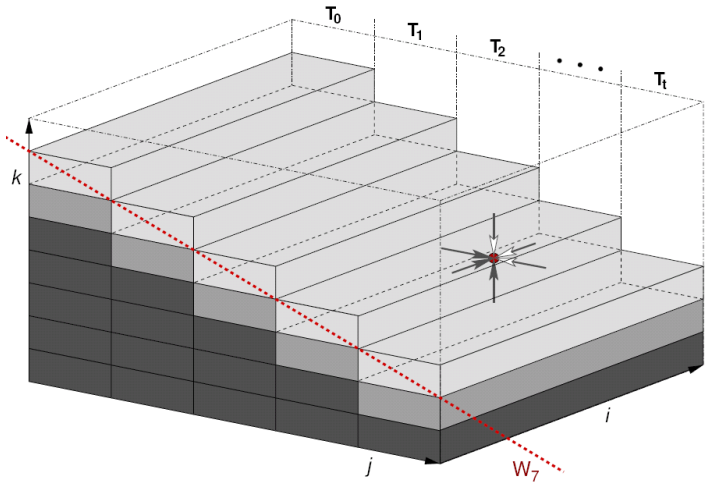
- Typical behavior for bandwidth-limited code!
- Roofline model is accurate in the saturated regime
- No scaling across sockets if only the main loop is parallel!
  - Fix by paralell first-touch initialization

Assignment 6, Task 2:

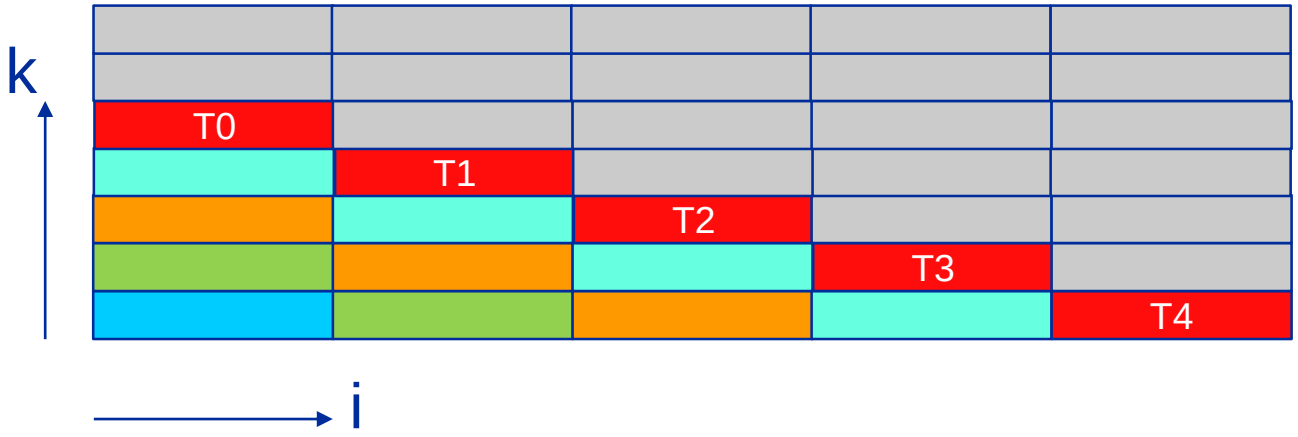
PPP for Gauss-Seidel smoother

Principle in 3D (“pipeline parallel processing”):



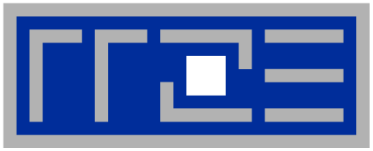


2D:



Assignment 6, Task 2:

PPP for Gauss-Seidel smoother

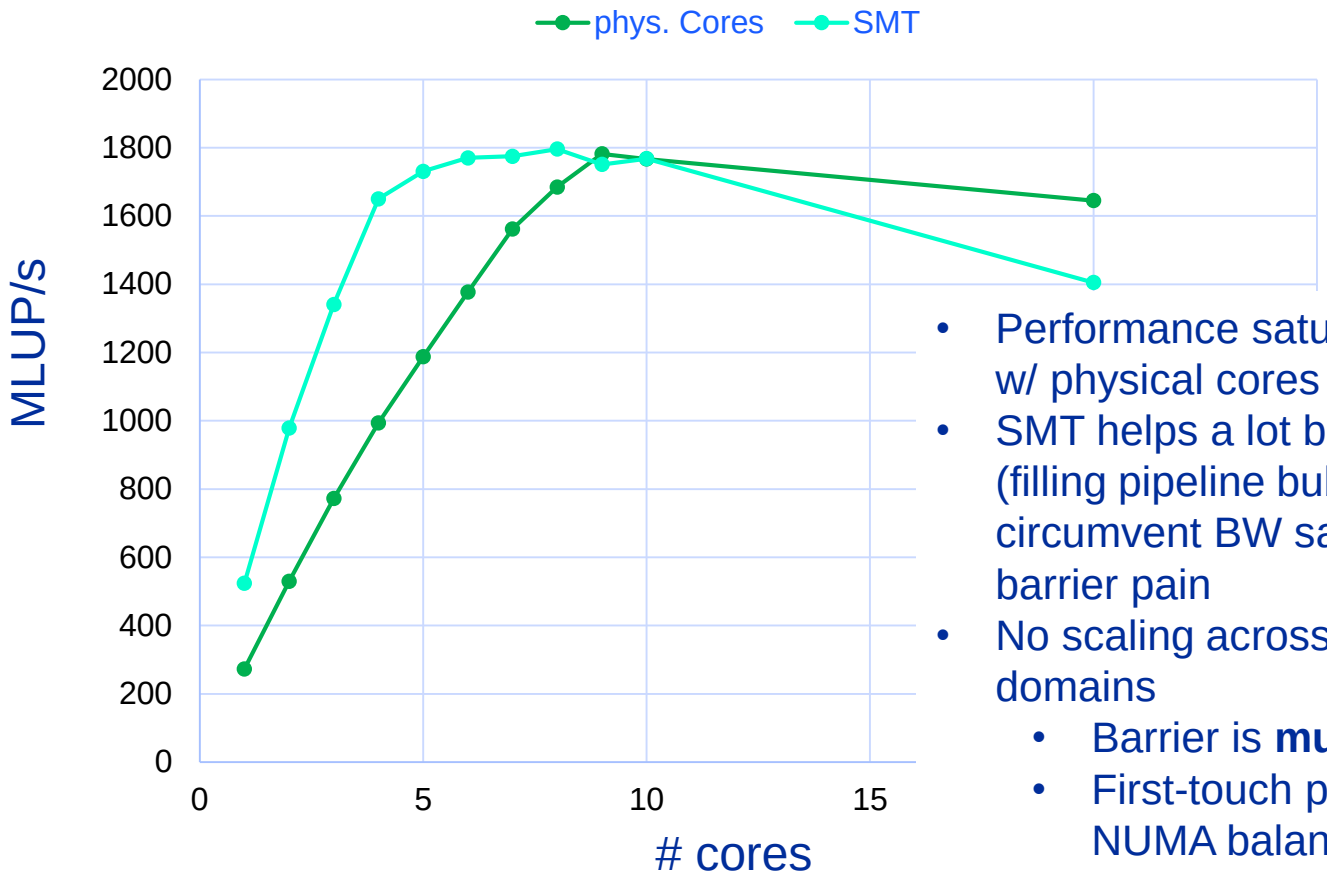


Pipelined parallel processing via OpenMP

```
!$OMP parallel private(nthreads,istart,iend,tid,kk,it,k,i)
  nthreads = omp_get_num_threads()
  tid = omp_get_thread_num()
  istart= (imax-1)/nthreads * tid +1
  iend  = istart+(imax-1)/nthreads-1
  do it=1,itmax
     do k=1,kmax-1+nthreads-1
        kk = k - tid
        if(kk .ge. 1 .and. kk .le. kmax-1) then
           do i=istart, iend
              phi(i,kk) = 0.25d0 * ( phi(i,kk-1)+
                phi(i+1,kk) +  phi(i,kk+1) + phi(i-1,kk))
           enddo
        endif
!$OMP barrier
     enddo ! k
  enddo !it
!$OMP end parallel
```

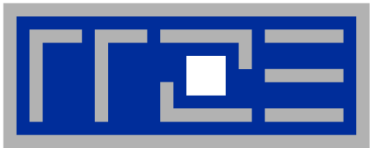
PPP 2D Gauss-Seidel measurements

- Emmy node @ 2.2 GHz, grid size 8000x8000

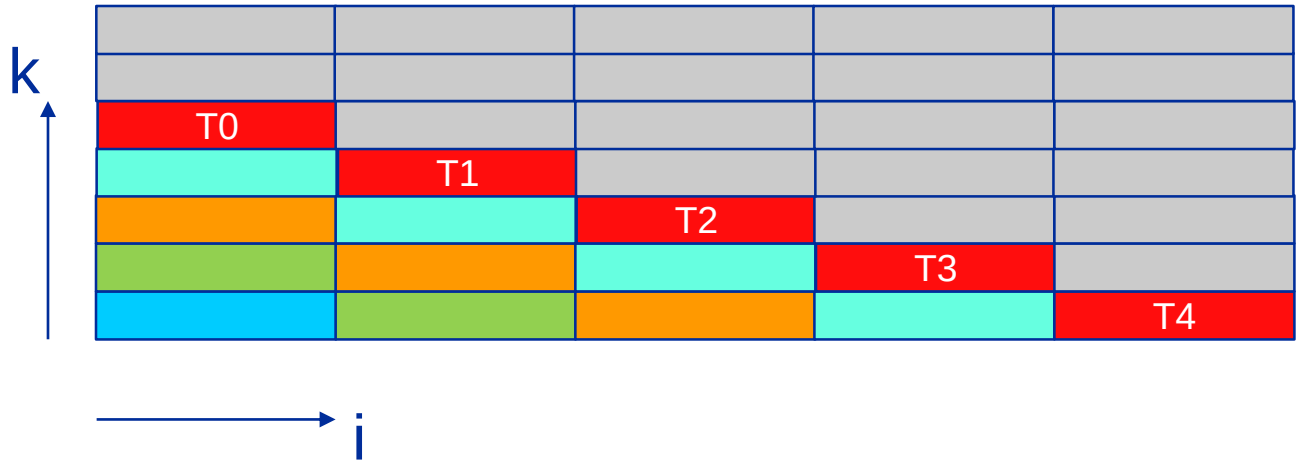


- Performance saturates just barely w/ physical cores only
- SMT helps a lot below saturation (filling pipeline bubbles) but cannot circumvent BW saturation or barrier pain
- No scaling across ccNUMA domains
  - Barrier is **much** more costly
  - First-touch placement? NUMA balancing?

PPP 2D Gauss-Seidel ccNUMA issues

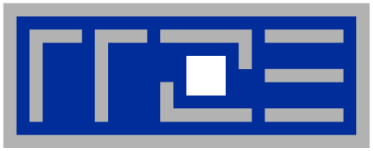


- Which loop to parallelize upon initialization for proper ccNUMA placement?



  - Steady state (after wind-up) has fixed i-block-to-thread mapping
  - → parallelize i loop upon init!
- But does it work for imax=8000 on Emmy?
  - No! ← 2 MiB page size on Emmy nodes
    (it would work with 4 KiB, sort of)
  - Last resort: static,1 parallelization of outer loop

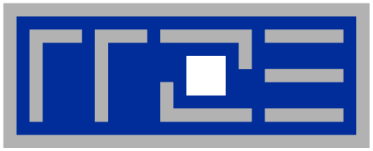
PPP 2D Gauss-Seidel ccNUMA issues



- Round-robin first touch initialization → „random“ placement

```
!$OMP PARALLEL DO schedule(static,1)
  do k=1,kmax-1
     do i=1,imax-1
        phi(i,k)=1.d0
     enddo
  enddo
!$OMP END PARALLEL DO
```

PPP 2D Gauss-Seidel Performance model



- Bandwidth-bound performance limit on one Emmy socket:

  Layer condition: 3 rows must fit in 12 MB of L3 → imax <= 521000

  B_c = 16 B/LUP (read & write each lattice node)
  b_S = 41 GB/s

  → b_S / B_c = 2560 MLUP/s

- Impact of 2000 cy OpenMP barrier @ 8000x8000

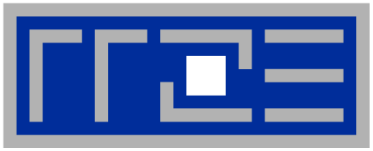
  Duration of one barrier-free update sweep (8000 – 2 LUPs):

  T_{bf} = (7998 LUPs / 2560 MLUP/s) * 2.2 Gcy/s = 6870 cy

  → performance reduction by factor of
  6870/(2000+6870) = 0.77 → 1980 MLUP/s

Assignment 6, Task 2:

Using the LIKWID Marker API



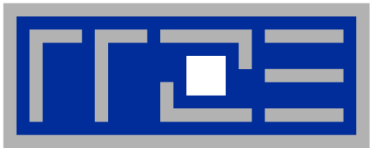
```
 call likwid_MarkerInit
 call likwid_MarkerRegisterRegion("ITER")
!...
 call likwid_MarkerStartRegion("ITER")
!$OMP parallel private(nthreads,istart,iend,tid,kk,it,k,i)
  nthreads = omp_get_num_threads()
  tid = omp_get_thread_num()
  istart= (imax-1)/nthreads * tid +1
  iend  = istart+(imax-1)/nthreads-1
  do it=1,itmax
! ... Sweep here
  enddo
!$OMP end parallel
call likwid_MarkerStopRegion("ITER")
!...
call likwid_MarkerClose
```

Optional – Reduces overhead for 1st marker call

Region for counting events

```
$ ifort -qopenmp ... $LIKWID_INC gs_opt_par.f90 timing.o \
   $LIKWID_LIB -llikwid
$ likwid-perfctr -g MEM -C S0:0-9 -m ./a.out
```

Assignment 6, Task 2:
Using the LIKWID Marker API



- **Output:**

```
[...]
Performance:    1839.7 MLUPs
Region ITER, Group 1: MEM
+-------------------+----------+
|    Region Info    |  Core 0  |
+-------------------+----------+
| RDTSC Runtime [s] | 5.213142 |
|     call count    |        4 |
+-------------------+----------+
[...]
+-----------------------------------+------------+
|               Metric              |   Core 0   |
+-----------------------------------+------------+
|        Runtime (RDTSC) [s]        |     5.2131 |
|        Runtime unhalted [s]       |     5.1748 |
|            Clock [MHz]            |  2200.0600 |
|                CPI                |     1.1604 |
|  Memory read bandwidth [MBytes/s] | 14808.2476 |
|  Memory read data volume [GBytes] |    77.1975 |
| Memory write bandwidth [MBytes/s] | 14762.9001 |
| Memory write data volume [GBytes] |    76.9611 |
|    Memory bandwidth [MBytes/s]    | 29571.1477 |
|    Memory data volume [GBytes]    |   154.1586 |
+-----------------------------------+------------+
```

$$B_C = \frac{29.57\ \text{GB/s}}{1.84\ \text{GLUP/s}} \approx 16.1\ \text{B/LUP}$$

Measured code balance