Performance Engineering
Case study: “Jacobi” stencil

The basics in two dimensions (2D)

Layer condition in 2D

From 2D to 3D

OpenMP parallelization strategies and layer condition in 3D

NT stores

Prof. Dr. G. Wellein[a,b] , Dr. G. Hager[a]
[a] Erlangen National High Performance Computing Center (NHR@FAU)
[b] Department für Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg, Sommersemester 2026

Case study: Jacobi stencil

The basics in two dimensions (2D)

Layer condition in 2D

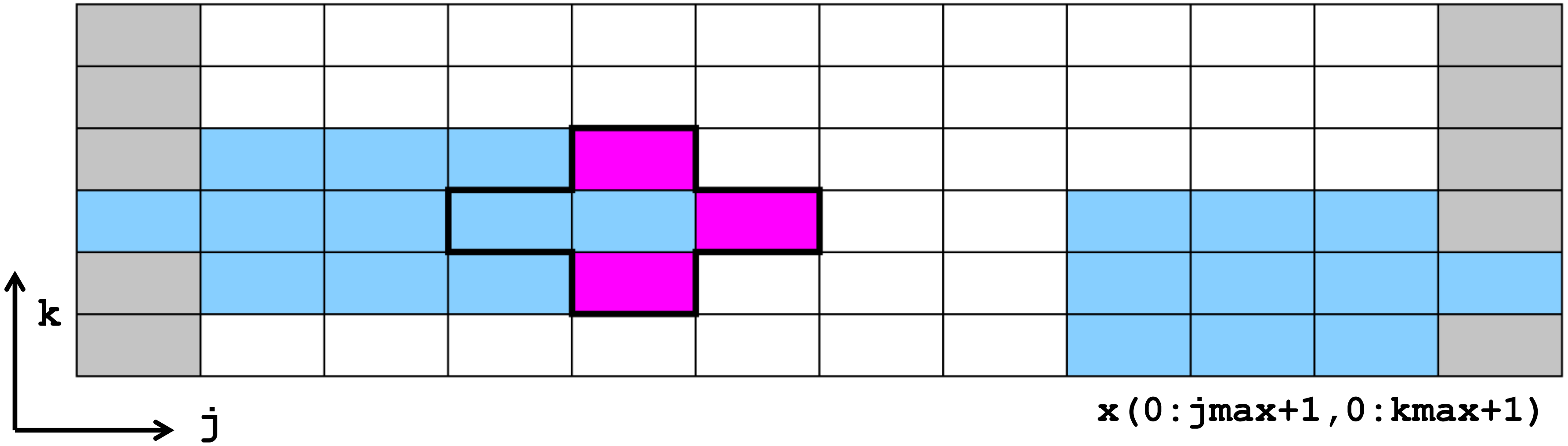
From 2D to 3D

OpenMP parallelization strategies and layer condition in 3D

NT stores

From 2D to 3D

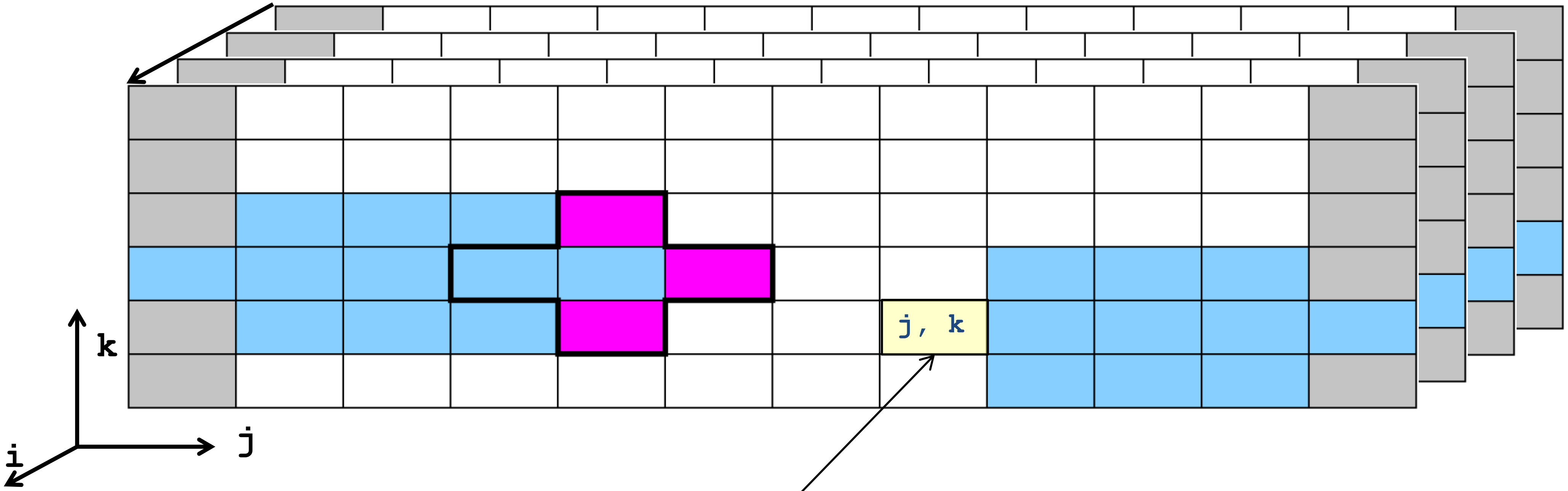
- 2D



- Towards 3D understanding
- Picture can be considered as 2D cut of 3D domain for (new) fixed `i`-coordinate:

`x(0:jmax+1,0:kmax+1)` → `x(i, 0:jmax+1,0:kmax+1)`

From 2D to 3D



- `x(0:imax+1, 0:jmax+1,0:kmax+1)` – Assume `i`-direction contiguous in main memory (Fortran notation)
- Stay at 2D picture and consider one cell of `j-k` plane as a contiguous slab of elements in `i`-direction: `x(0:imax,j,k)`

Layer condition: From 2D 5-pt to 3D 7-pt Jacobi-type stencil

```
3*jmax * 8B < CacheSize/2
```

```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (x(j-1,k) + x(j+1,k)  &
                    +  x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

double precision

B_C = 24 B / LUP

```
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = const * (x(i-1,j,k) + x(i+1,j,k)
                       +  x(i,j-1,k) + x(i,j+1,k)  &
                       +  x(i,j,k-1) + x(i,j,k+1)  )
    enddo
  enddo
enddo
```

```
3*jmax *imax * 8B < CacheSize/2
```

double precision

B_C = 24 B / LUP

3D 7-pt Jacobi-type Stencil (sequential)

"Layer condition"
`3*jmax*imax*8B < CS/2`

"Layer condition" OK →
5 accesses to `x()` served by cache

```
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) =const. *(x(i-1,j,k)   +x(i+1,j,k)  &
                      + x(i,j-1,k)    +x(i,j+1,k)  &
                      + x(i,j,k-1)    +x(i,j,k+1)  )
    enddo
  enddo
enddo
```

Question: Does parallelization/multi-threading change the layer condition?

Case study: Jacobi stencil

The basics in two dimensions (2D)

Layer condition in 2D

From 2D to 3D

OpenMP parallelization strategies and layer condition in 3D

NT stores

Jacobi Stencil – OpenMP parallelization (I)

```
!$OMP PARALLEL DO SCHEDULE(STATIC)
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = 1/6.    *(x(i-1,j,k)      +x(i+1,j,k)  &
                         + x(i,j-1,k)      +x(i,j+1,k)
                         + x(i,j,k-1)      +x(i,j,k+1) )
    enddo
  enddo
enddo
```

Basic guideline:
Parallelize outermost loop

SCHEDULE(STATIC)
Equally large chunks in k-direction

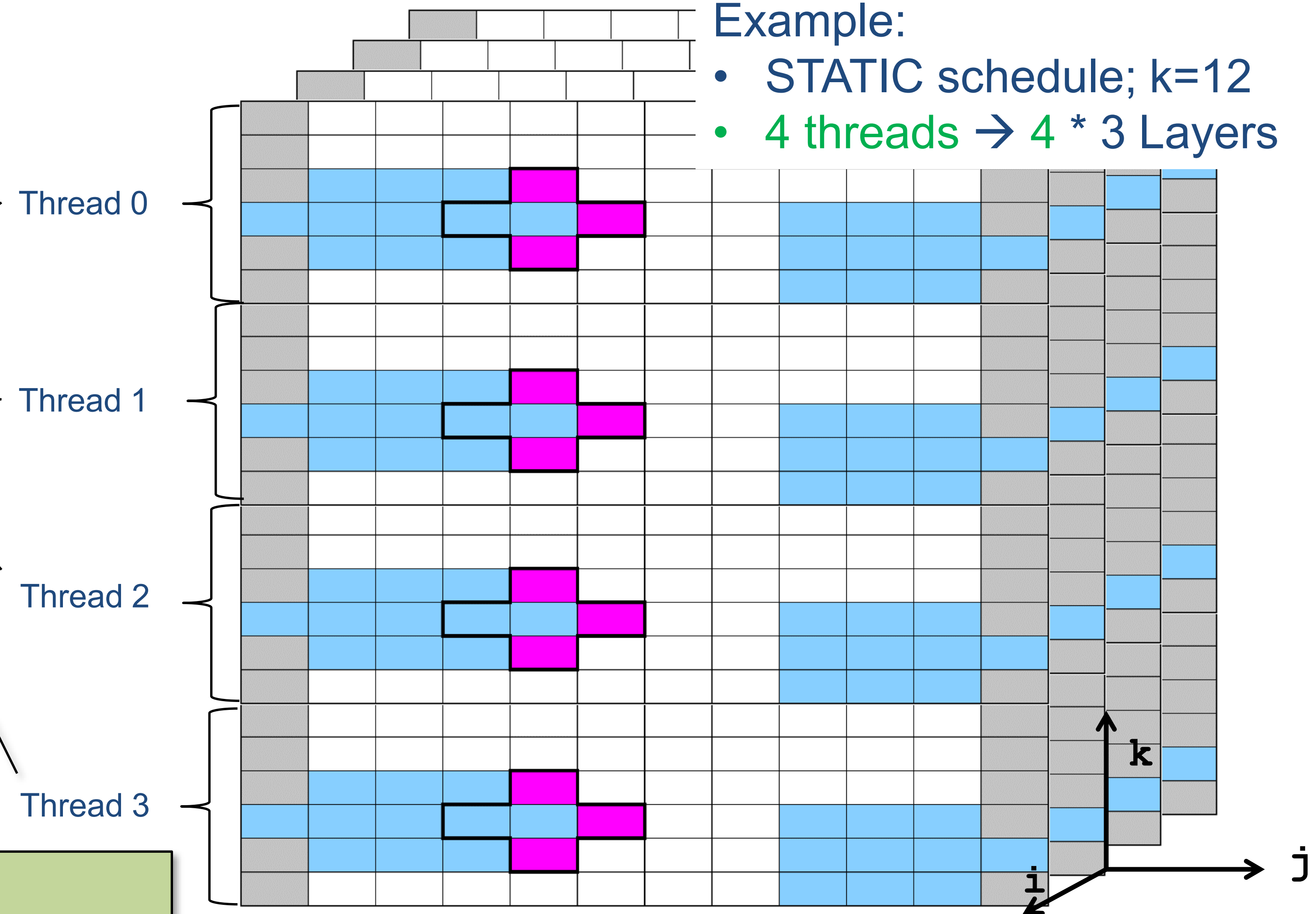
Jacobi Stencil – OpenMP parallelization (I)

Equally large chunks per thread in k-direction

Layer condition" to be fulfilled by each thread

Example:

- STATIC schedule; k=12
- 4 threads → 4 * 3 Layers



"Layer condition":

`nthreads *3*jmax*imax*8B < CS/2`

Note: Threads do not need to be in sync – they only sync at the end of the sweep

Jacobi Stencil – OpenMP parallelization (I)

Impact of parallelizationon **C**ache**S**ize: core-local vs. shared caches

- **CS_t** is the available cache size for holding data when using **`nthreads`** (=cores)

"Layer condition" (LC): `nthreads *3*jmax*imax*8B < CSt/2`

- If CS is **shared cache** (e.g. L3): **CS_t = constant**
  → LC more stringent at higher thread counts

`L3-CacheSize=25 MB`

1 thread: `imax=jmax < 720`
10 threads: `imax=jmax < 230`

- If CS is core-local cache (e.g. L2): **CS_t = constant** * **`nthreads`**
  → LC independent of thread count

`L2-CacheSize=256 KB`

1 thread: `imax=jmax < 70`
10 threads: `imax=jmax < 70`

Jacobi Stencil – OpenMP parallelization (I)

"Layer condition": `nthreads *3*jmax*imax*8B < CS_t/2`

```
!$OMP PARALLEL DO SCHEDULE(STATIC)
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = 1/6. *(x(i-1,j,k)   +x(i+1,j,k) &
                      + x(i,j-1,k)   +x(i,j+1,k) &
                      + x(i,j,k-1)   +x(i,j,k+1) )
    enddo
  enddo
enddo
```

double precision

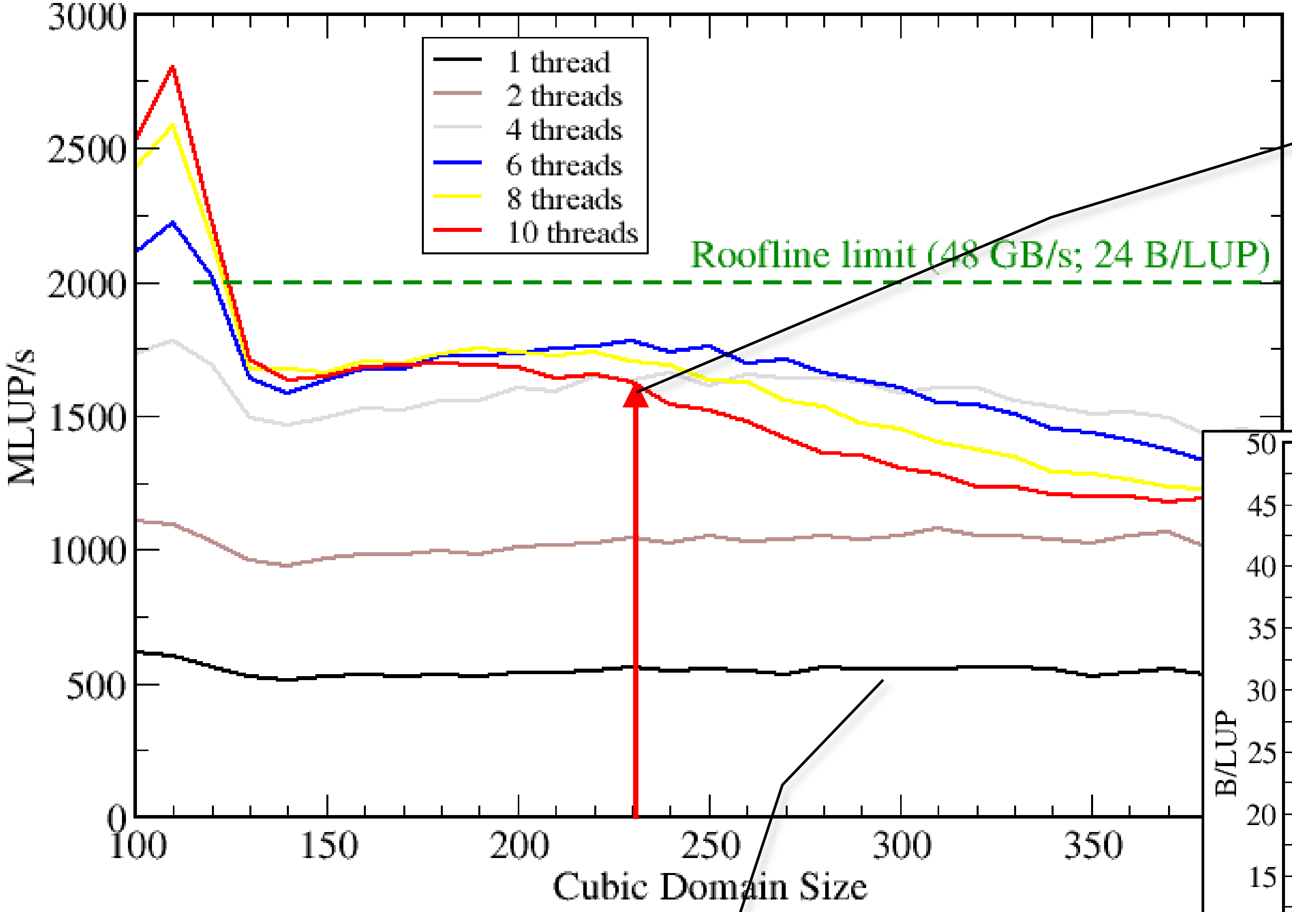
B_C = 24 B / LUP

Roofline model:
$P = b_S / B_C^{mem}$

Intel® Xeon® Processor E5-2690 v2
10 cores@3 GHz
CacheSize = 25 MB (L3; shared)
$b_S = 48 \text{ GB/s}$

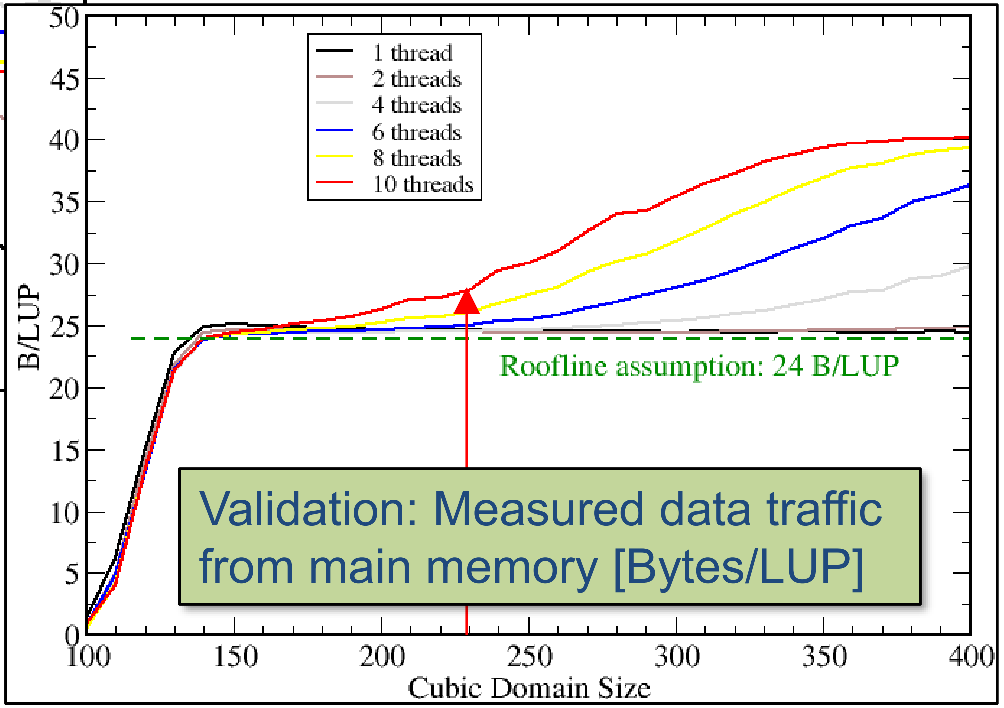
Best performance:
$P = 2000 \text{ MLUP/s}$

Jacobi Stencil – OpenMP parallelization (I)



10 threads: performance drops around **`imax=230`**

1 thread: Layer condition OK – but can not saturate bandwidth



Validation: Measured data traffic from main memory [Bytes/LUP]

Jacobi Stencil – OpenMP parallelization (I)

Original “Layer condition” does not hold: `nthreads*3*jmax*imax*8B` > $CS_t/2$

```
!$OMP PARALLEL DO SCHEDULE(STATIC)
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = 1/6.    *(x(i-1,j,k)      +x(i+1,j,k)  &
                         + x(i,j-1,k)      +x(i,j+1,k)
                         + x(i,j,k-1)      +x(i,j,k+1) )
    enddo
  enddo
enddo
```

But assume: `nthreads*3*imax*8B < CS/2`

```
 (8+8) B/LUP  for y()  (ST+WA)
+   8  B/LUP  for x(i,j,k+1)
+   8  B/LUP  for x(i,j+1,k)
+   8  B/LUP  for x(i,j,k-1)
```

→ $B_C = 40 \text{ B/LUP}$

→ $P = 1200 \text{ MLUP/s}$

Jacobi Stencil – OpenMP & spatial blocking

```
do jb=1,jmax,jblock ! Assume jmax is multiple of jblock

!$OMP PARALLEL DO SCHEDULE(STATIC)
  do k=1,kmax
    do j=jb, (jb+jblock-1) !  Loop length jblock
      do i=1,imax
        y(i,j,k) = 1/6. *(x(i-1,j,k) +x(i+1,j,k) &
                   +     x(i,j-1,k) +x(i,j+1,k)
                   +     x(i,j,k-1) +x(i,j,k+1))
      enddo
    enddo
  enddo
enddo
```

"Layer condition" (j-Blocking))

`nthreads*3*jblock*imax*8B < CS_t/2`

Ensure layer condition by choosing `jblock` approriately (Cubic Domains):

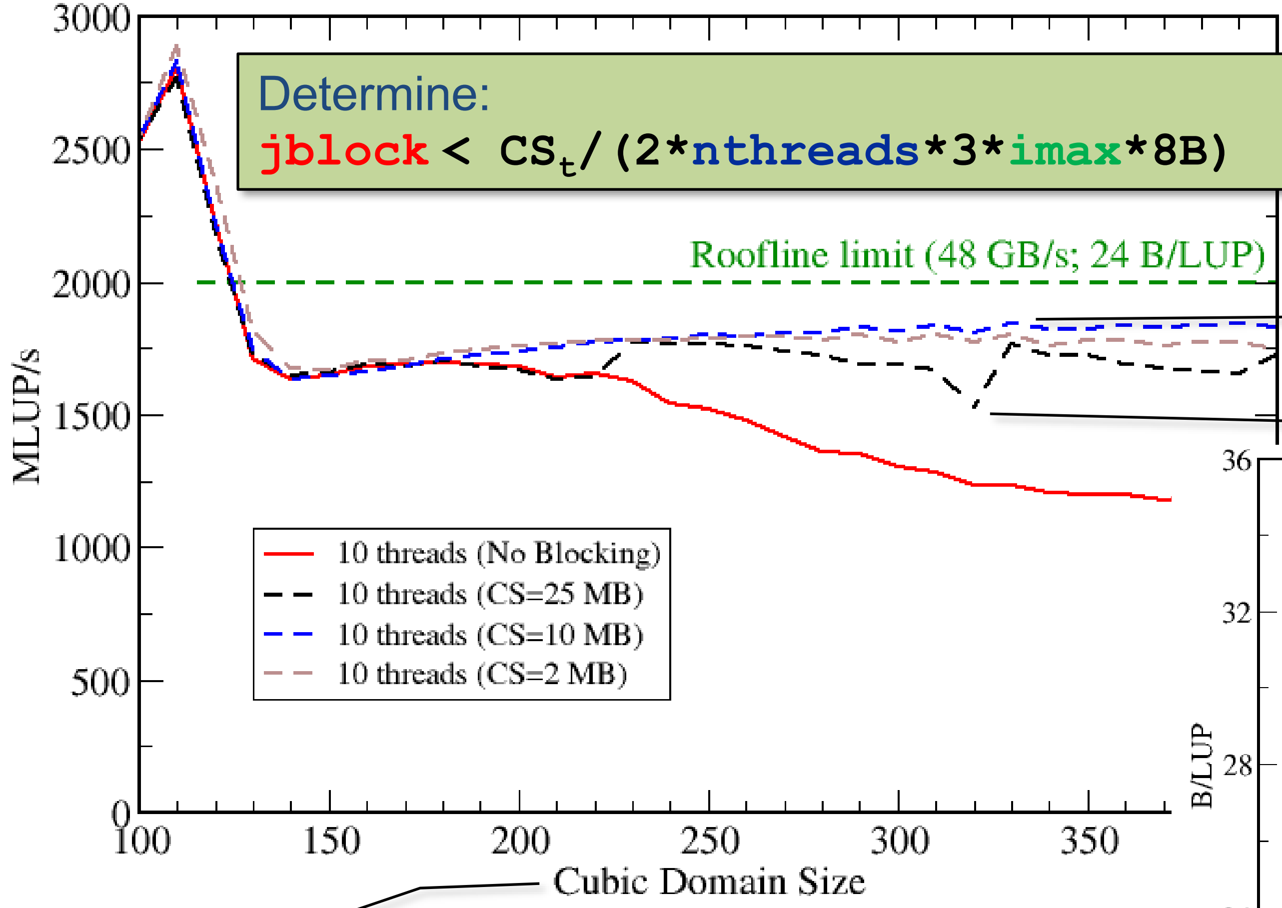
`jblock < CS_t/(imax * nthreads * 48B )`

→ Back to prediction of $P = 2000\ MLUP/s$

Jacobi Stencil – OpenMP & spatial blocking

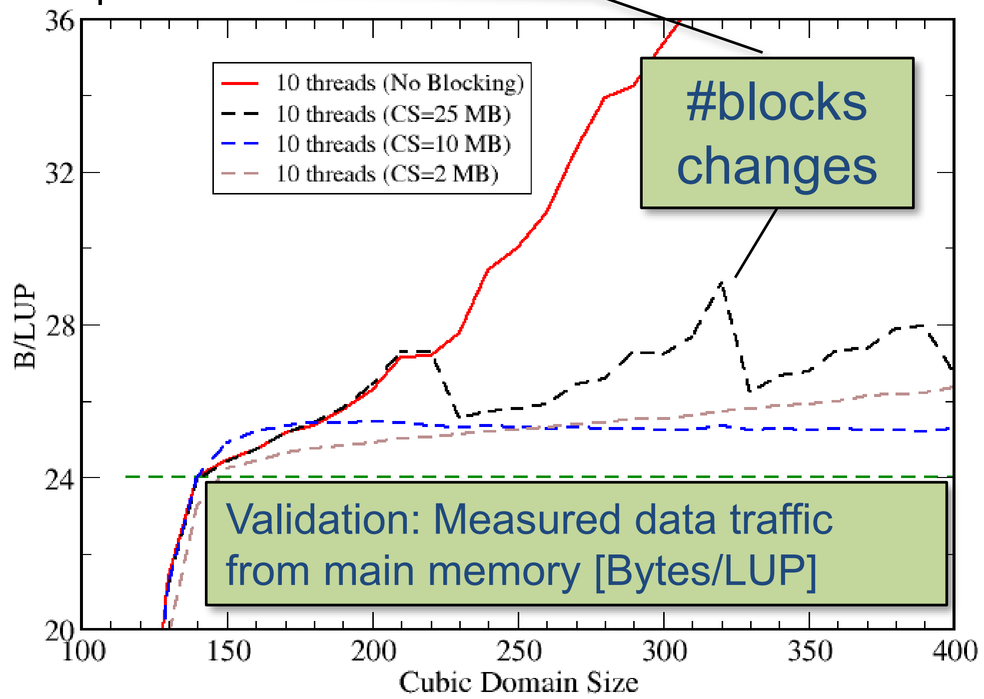
Determine:

`jblock < CS_t/(2*nthreads*3*imax*8B)`



CS_t=10 MB:
~ 90+ % roofline limit

`imax = jmax = kmax`



#blocks changes

Validation: Measured data traffic from main memory [Bytes/LUP]

Impact of blocking factor jblock

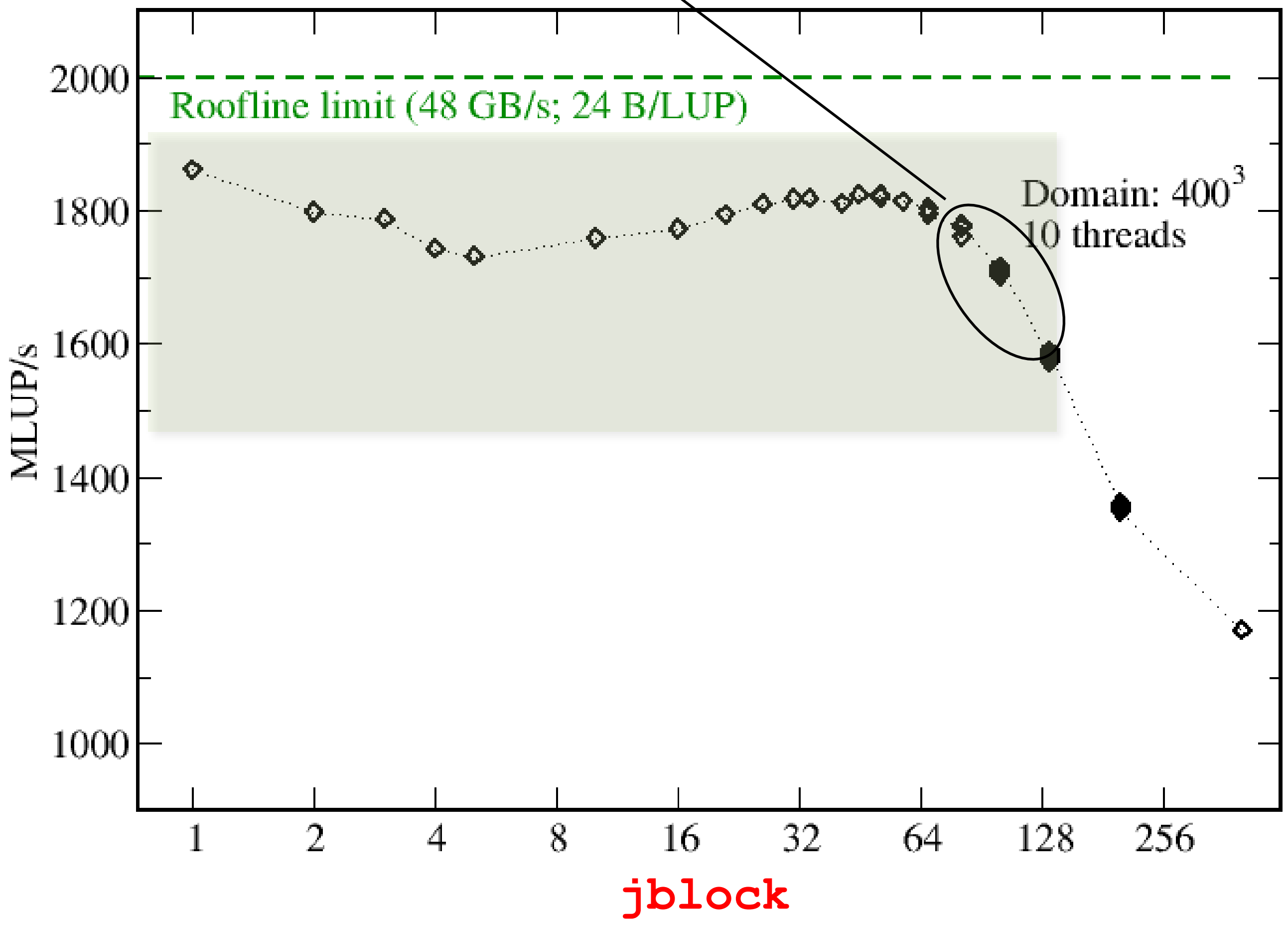
Layer condition estimates appropriate jblock:
`jblock < CS_t/(2*nthreads*3*imax*8B)`

CS_t=25 MB
nthreads=10
imax=400

jblock < 130

jblock=32,…,64
useful choices

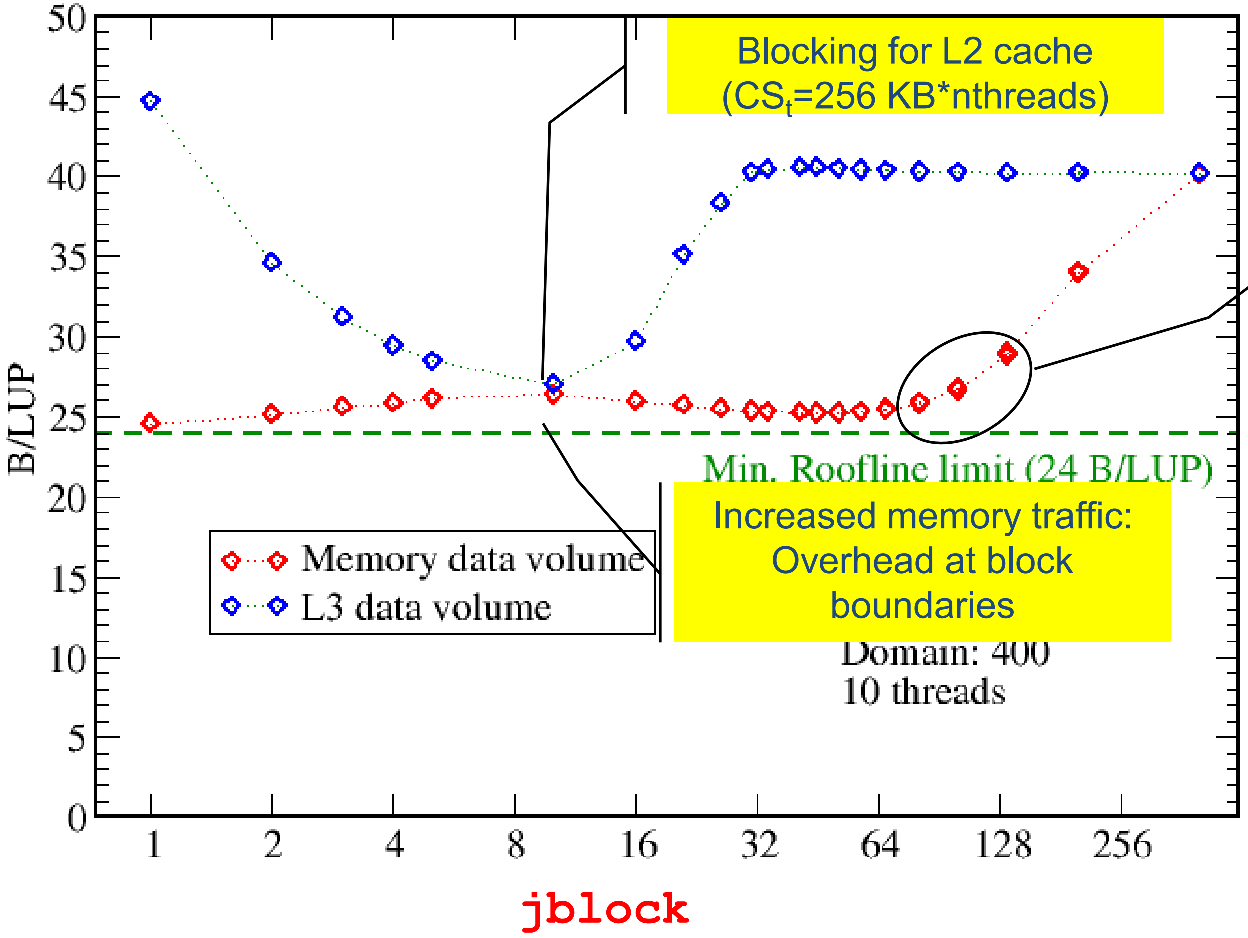
Layer condition a bit too optimistic



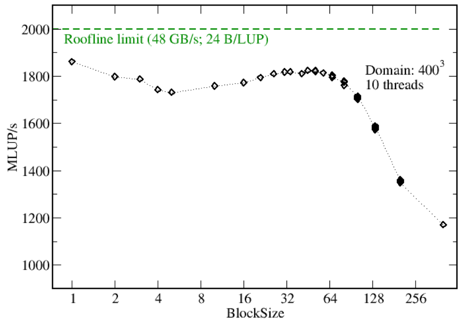
Impact of jblock: Data traffic from L3 & main memory

Layer condition estimates appropriate jblock:
`jblock < CS_t/(2*nthreads*3*imax*8B)`

jblock < 130



Layer condition a bit too optimistic



Jacobi Stencil – what about j-loop parallelization

```
!$OMP PARALLEL PRIVATE(k)
do k=1,kmax
!$OMP DO SCHEDULE(STATIC)
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = 1/6. *(x(i-1,j,k) +x(i+1,j,k) &
                      + x(i,j-1,k) +x(i,j+1,k) +x(i,j,k-1) +x(i,j,k+1) )
    enddo
  enddo
!$OMP END DO
enddo
!$OMP END PARALLEL
```

All threads work on the same 3 k-layers at the same time

"Layer condition" (j-loop parallel; schedule=static)
$3*jmax*imax*8B < CS_t/2$

Layer condition is independent of number of threads (`imax<720` for test system)

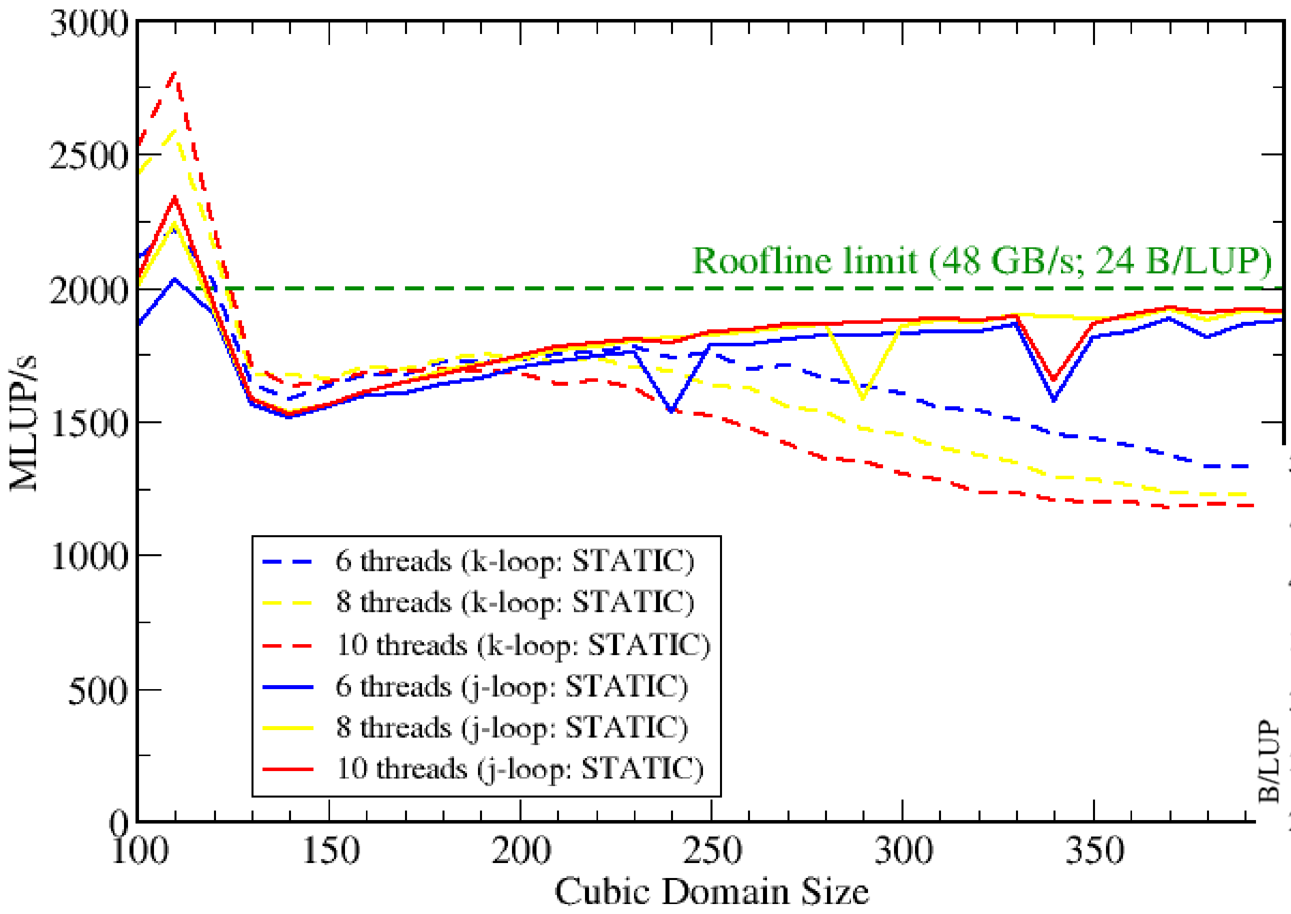
Impact of parallelization overhead?!

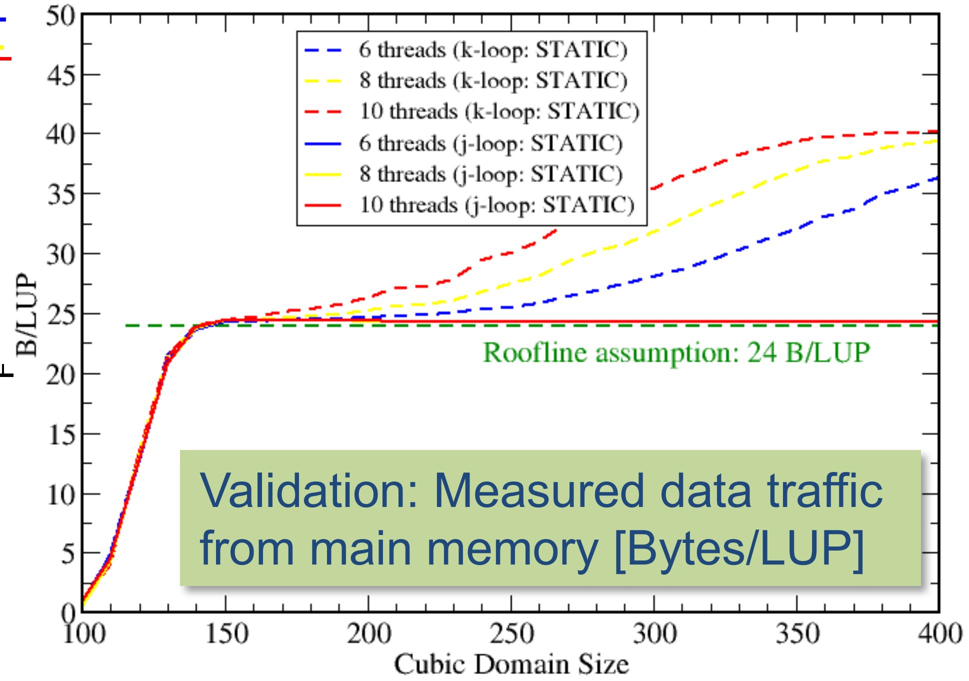
- 1 barrier each `jmax * imax` LUPs
- Costs / barrier: approx. `1 500 cycles`
- Computational time / barrier: (`jmax * imax LUPs)/maxMLUPs`

  `e.g. jmax=imax=200 & maxMLUPs=2000 MLUPs` → $20 * 10^{-6}$ s

  → `60 000 cycles`

Jacobi Stencil – what about j-loop parallelization





Performance will drop at domain sizes ~500,..,700

Case study: Jacobi stencil

The basics in two dimensions (2D)

Layer condition in 2D

From 2D to 3D

OpenMP parallelization strategies and layer condition in 3D

NT stores

Jacobi Stencil – can we further improve?

```
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) =const. *(x(i-1,j,k)  +x(i+1,j,k)  &
                      + x(i,j-1,k)   +x(i,j+1,k)  &
                      + x(i,j,k-1)   +x(i,j,k+1)  )
    enddo
  enddo
enddo
```

"Layer condition" OK →
5 accesses to **x()** served by cache

Total data transfer / LUP:

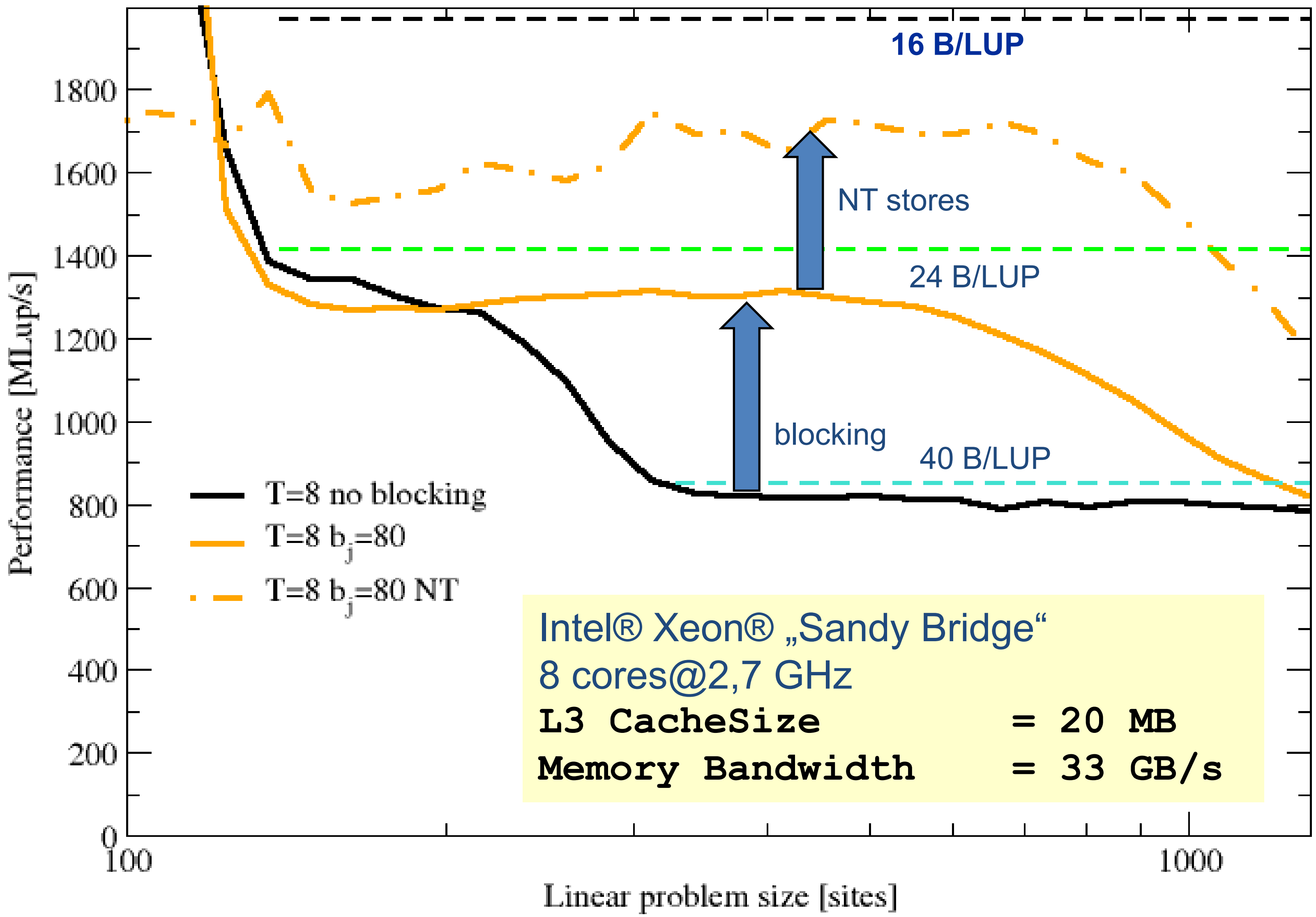
```
 (8+8) B/LUP for y()  (ST+Write Allocate)
+    8 B/LUP for x(i,j,k+1)
→ 24   B/LUP
```

Use NT-stores to avoid "Write Allocate"

Total data transfer / LUP:

```
     8  B/LUP  for y()  (NT-STore)
+    8  B/LUP  for x(i,j,k+1)
→ 16    B/LUP
```

Jacobi Stencil – Blocking + NT-stores



Conclusions from the Jacobi example

- Data transfer analysis is key to structured performance modeling and optimization
- Avoiding slow data paths == re-establishing the most favorable layer condition
- Optimal blocking factor for spatial blocking can be estimated
  - Be guided by the cache size the layer condition → does not depend on outer loop!
  - Blocking: overhead at block boundaries → Largest cache first choice
  - No need for exhaustive scan of “optimization space”
- Performance optimization by re-establishing the layer condition was guided and validated by (roofline-like) model
- Modeling and optimizing “Jacobi” stencil is blueprint for many stencil(-like) kernels.

Open topics – layer condition: 3D (outer loop parallel)

Derived for
OMP_SCHEDULE=STATIC
→ Does it change for other scheduling strategies?

$$\texttt{nthreads} * 3 * \underbrace{\texttt{jmax} * \texttt{imax}} * \, 8B < CS_t/2$$

What about "square blocking":
jmax → jblock &
imax → iblock ?

What about i-blocking:
imax → iblock ?

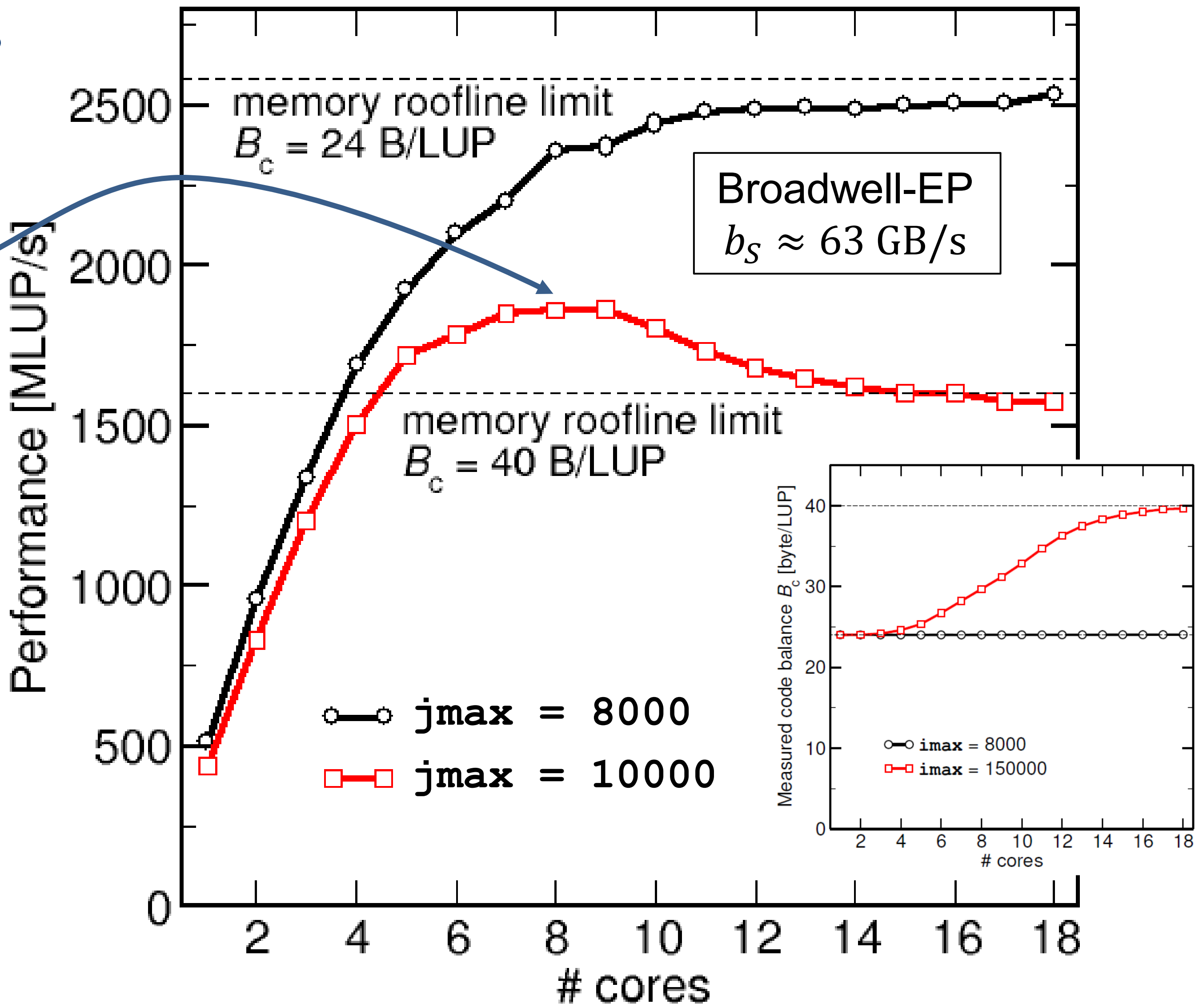
OpenMP parallelization and blocking for a shared cache

Layer conditions make for interesting effects

- Less and less shared cache available per thread as #threads goes up
- LC may break "along the way"

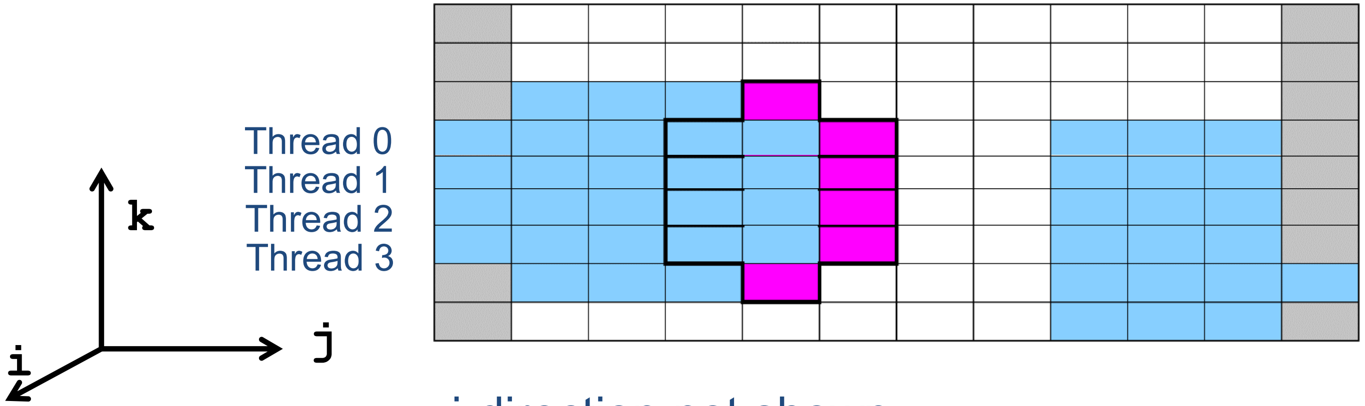
- Solutions
  1. Choose small enough block or domain size
     - Layers either small enough to fit in core-private caches *or*
     - Shared cache big enough to hold all layers for all threads
  2. Adaptive blocking for shared cache:

$$\texttt{jblock} = \frac{C}{\#threads \times 48\ \mathrm{byte}}$$



Open topics – layer condition & outer loop parallel w/ STATIC,1 schedule

```
!$OMP PARALLEL DO SCHEDULE(STATIC,1)
do k=1,kmax
  do j=1,jmax
    do i=1,imax
      y(i,j,k) = 1/6.  *(x(i-1,j,k)   +x(i+1,j,k) &
                       + x(i,j-1,k)   +x(i,j+1,k)
                       + x(i,j,k-1)   +x(i,j,k+1) )
    enddo
  enddo
enddo
```



i-direction not shown

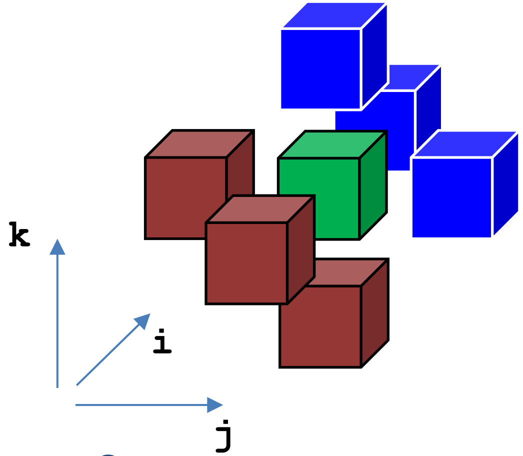
What is the relevant layer condition here???

Case study: Parallelizing a Gauss-Seidel Solver

3D matrix-free Gauss-Seidel smoother

- Matrix-free iterative solver for $Ax = b$

$$x_i^{(k)} = \frac{1}{a_{ii}}\left(-\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} + b_i\right)$$



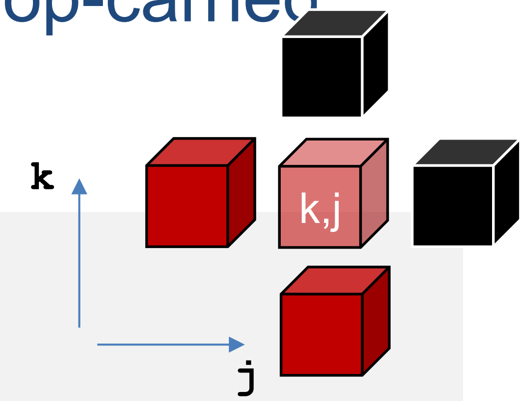
- Here used for Dirichlet boundary value (PDE) problem $\Delta x = 0$

```
for(it=0; it<itmax; ++it) { // or convergence check
  for(k=1; k<kmax-1; ++k) {
    for(j=1; j<jmax-1; ++j) {
      for(i=1; i<imax-1; ++i) {
        x[k][j][i] = ( x[k][j][i-1]   + x[k][j][i+1]
                     + x[k][j-1][i]   + x[k][j+1][i]
                     + x[k-1][j][i]   + x[k+1][j][i]) /6.0;
} } } }
```

"new data" "old data"

OpenMP parallelization?

- Naïve OpenMP loop parallelzation impossible due to loop-carried dependency on all spatial loop levels

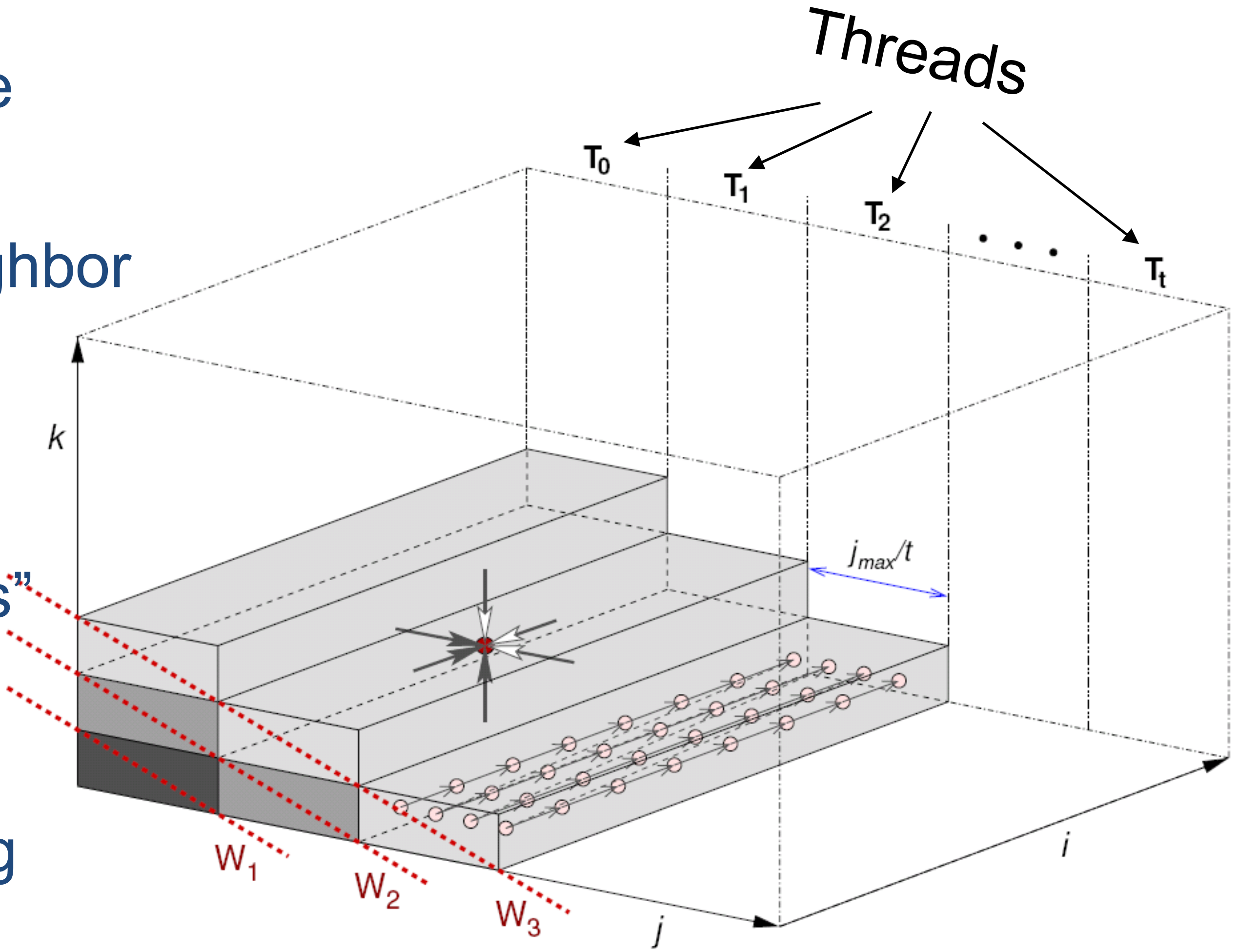



```
for(t=0; t<itmax; ++t) {
  for(k=1; k<kmax-1; ++k) {
    for(j=1; j<jmax-1; ++j) {
      for(i=1; i<imax-1; ++i) {
        x[k][j][i] = ( x[k][j][i-1] + x[k][j][i+1]
                     + x[k][j-1][i] + x[k][j+1][i]
                     + x[k-1][j][i] + x[k+1][j][i])/6.0;
} } } }
```

- Can we solve this in parallel but still keep the dependencies intact?

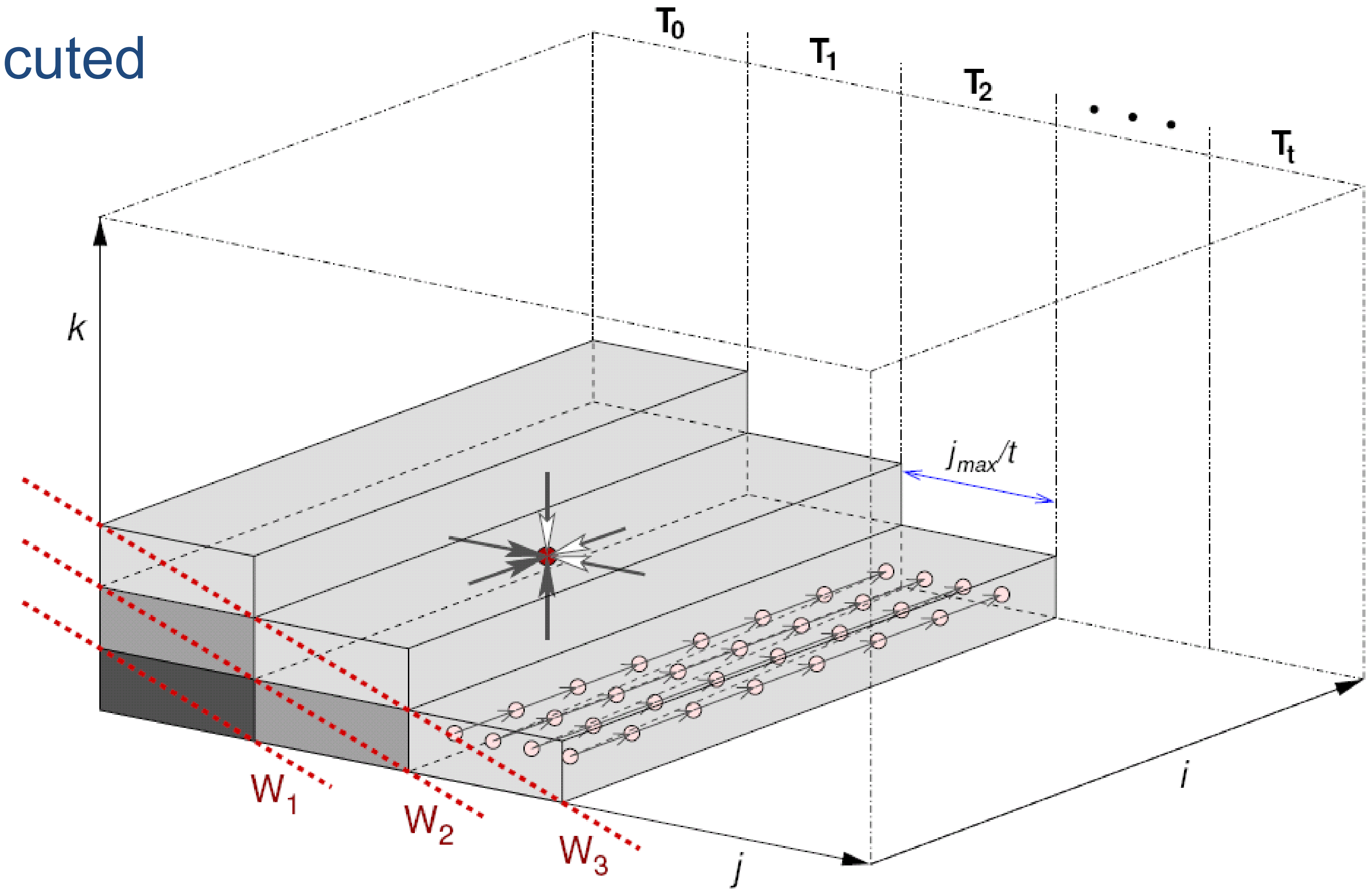
Idea: wavefront parallelization

- Parallelization approach
  - Middle (j) loop is parallel
  - Outer dimension: wavefront scheme
  - Block can be updated if bottom neighbor (same threadID) and left neighbor (threadID-1) are done
  - W_i: independent blocks, “wavefronts”
  - After each wavefront: synchronization to maintain ordering



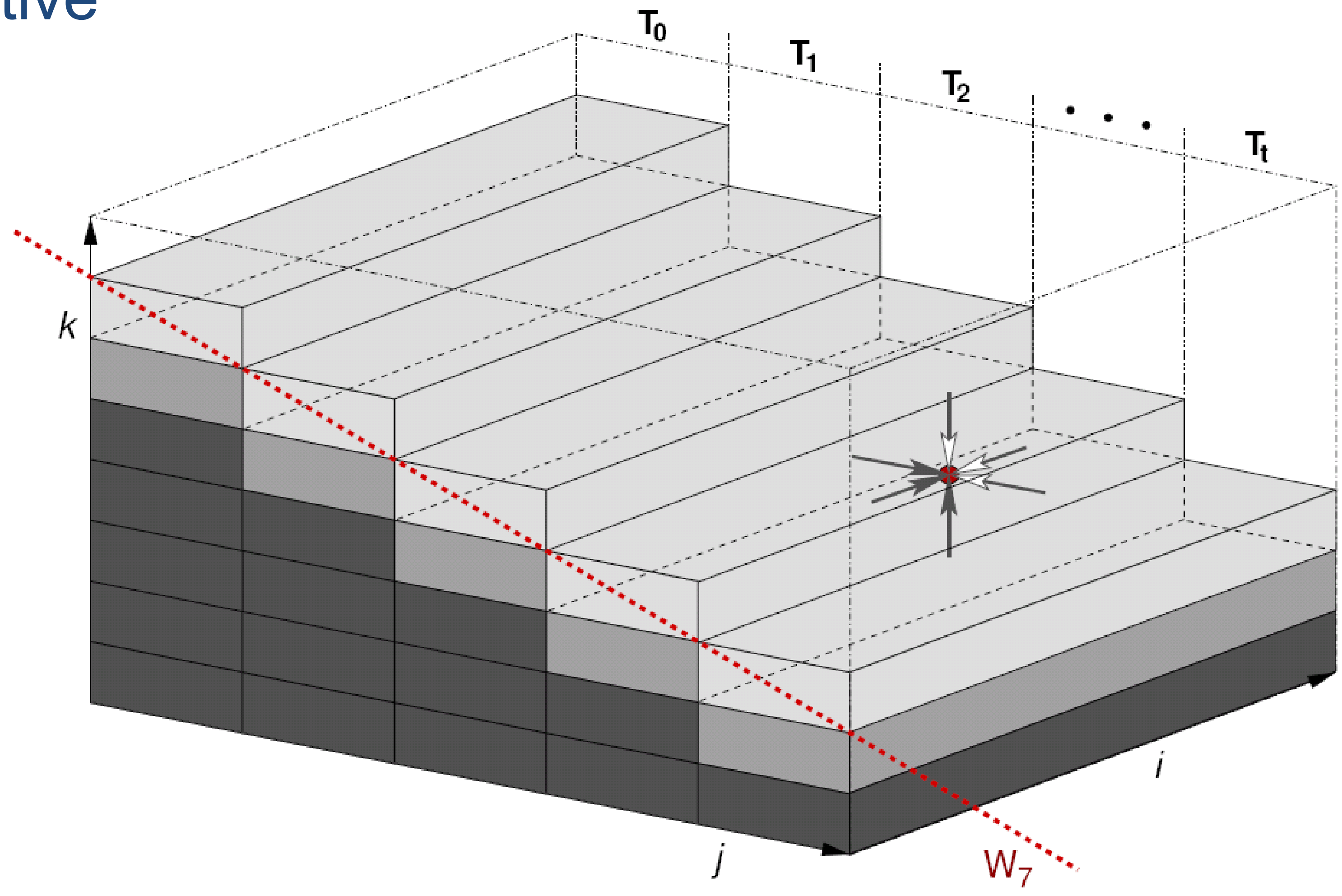
Wavefront parallelization

- Wind-up phase
  - Not all threads active
  - Each wavefront (W_i) is executed by i threads concurrently



Wavefront parallelization

- “Full pipeline”: All threads active



- Wind-down phase starts after T_0 has completed its k loop (not shown)

Wavefront parallelization with OpenMP in 3D

```
#pragma omp parallel private(nth,istart,iend,tid,kk,it,k,i){
  nth = omp_get_num_threads();
  tid = omp_get_thread_num();
  jstart= (jmax-2)/nth * tid + 1;
  jend  = (tid==nth-1 ? jmax-2 : jstart+(jmax-2)/nth-1);
  for(t=0; t<itmax; ++t) {
    for(k=1; k<kmax-1+nth-1; ++k) {
      kk = k - tid;
      if(kk>=1 && kk<kmax-1) {
        for(j=jstart; j<=jend; ++j) {
          for(i=1; i<kmax-1; ++i) {
            x[kk][j][i] = ( x[kk][j][i-1] + x[kk][j][i+1]
                          + x[kk][j-1][i] + x[kk][j+1][i]
                          + x[kk-1][j][i] + x[kk+1][j][i])/6.0;
          }
        }
      }
      #pragma omp barrier
    }
  }
}
```

Chop **j** loop into **nthreads** chunks

Wind-up/-down

Wavefront sync

Wavefront parallelization – open questions

- Global barrier per middle loop sweep (i.e., **`kmax-2`** barriers overall)
  - Remedy?

Is there a global performance limit, i.e. minimum balance?

- Minimum data traffic:
  → Update whole array **`x(0:imax+1,0:jmax+1,0:kmax+1)`** once
  → 16 byte per element (read & write)
  → Minimum data traffic: **`16*imax*jmax*kmax`** byte
- Work:
  → **`imax*jmax*kmax`** Lattice Site Updates (LUPs)

→ B_C= 16 byte/LUP