



Optimizing for Multicores

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se



Outline of these lectures

1. Processor implementations
2. Caches and memory system
3. Multiprocessors
4. HW optimizations
5. Multicore processors
6. **SW optimizations**



Optimizing for the memory system: What is the potential gain?

- Latency difference L1\$ and mem: $\sim 50x$
- Bandwidth difference L1\$ and mem: $\sim 20x$
- Execute from L1\$ instead from mem \implies 50-150x improvement
- At least a factor 2-4x is within reach



Optimizing for cache performance

- Keep the active footprint small
- Use the entire cache line once it has been brought into the cache
- Fetch a cache line prior to its usage
- Let the CPU that already has the data in its cache do the job
- ...



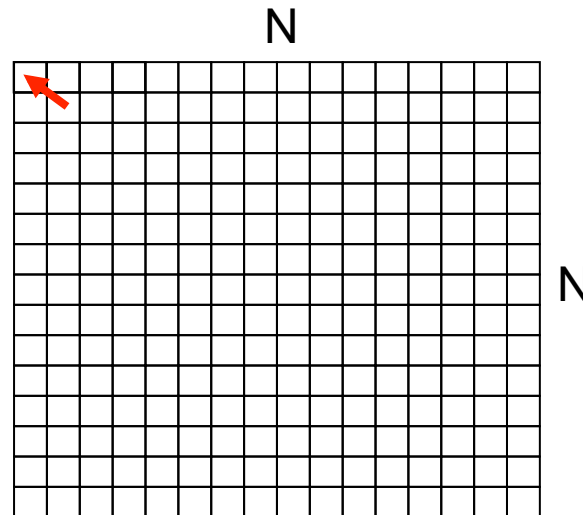
Final cache lingo slide

- **Miss ratio:** What is the likelihood that a memory access will miss in a cache?
- **Miss rate:** D:0 per time unit, e.g. per-second, per-1000-instructions
- **Fetch ratio/rate:** What is the likelihood that a memory access will cause a fetch to the cache [including HW prefetching]
- **Fetch utilization:** What fraction of a cacheline was used before it got evicted
- **Writeback utilization:** What fraction of a cacheline written back to memory contains dirty data
- **Communication utilization:** What fraction of a communicated cacheline is ever used?



What can go Wrong? A Simple Example...

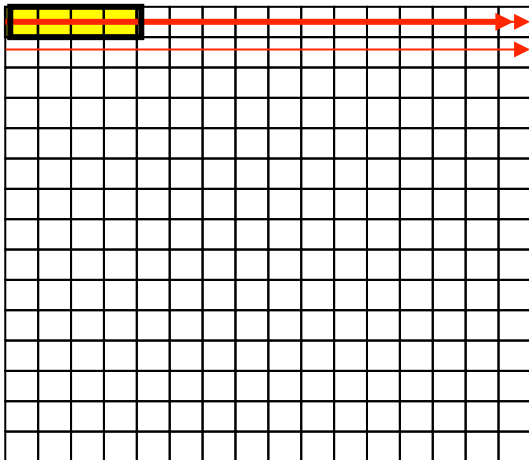
Perform a diagonal copy 10 times



Example: Loop order

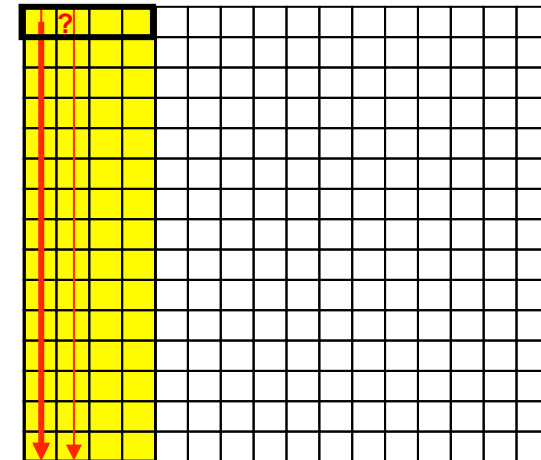
//Optimized Example A

```
for (i=1; i<N; i++) {  
  for (j=1; j<N; j++) {  
    A[i][j]= A[i+1][j+1];  
  }  
}
```



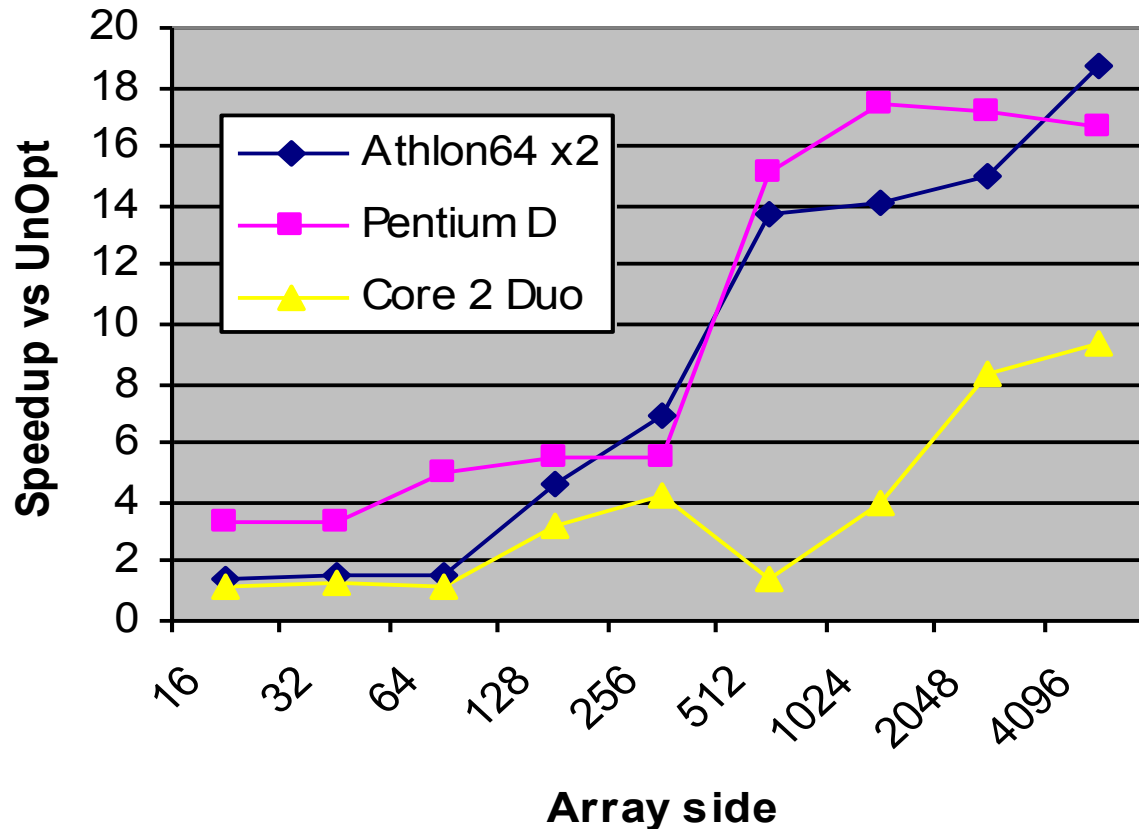
//Unoptimized Example A

```
for (j=1; j<N; j++) {  
  for (i=1; i<N; i++) {  
    A[i][j] = A[i+1][j+1];  
  }  
}
```



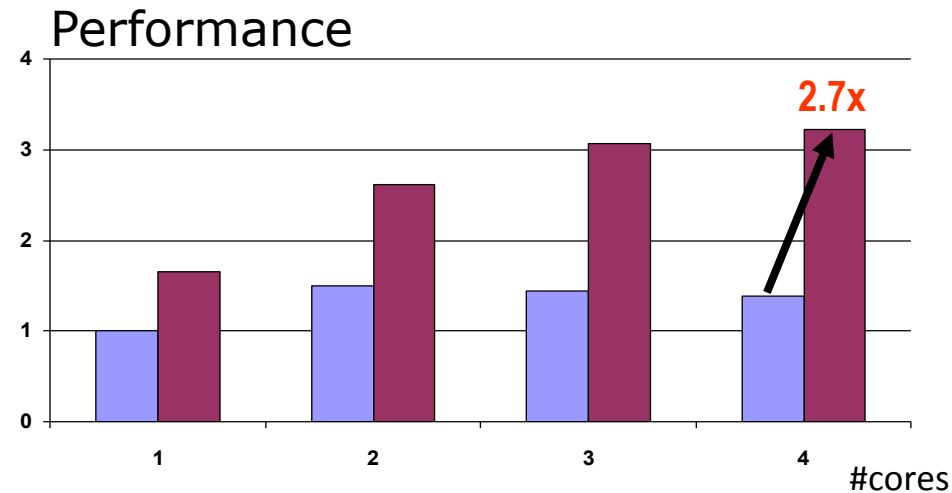


Performance Difference: Loop order





Example 1: LBM



App: LBM

Optimization can be rewarding, but costly...

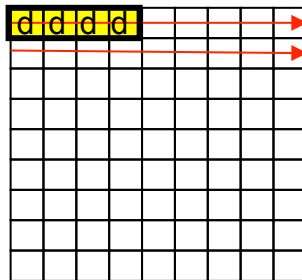
- Require expert knowledge about MC and architecture
- Weeks of wading through performance data

→ This fix required one line of code to change

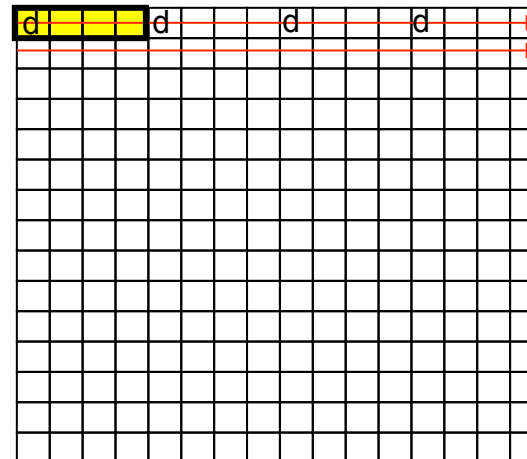
SW Optimizations 9

Example: Sparse data usage

```
//Optimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A_d[i][j]= A_d[i-1][j-1];
  }
}
```



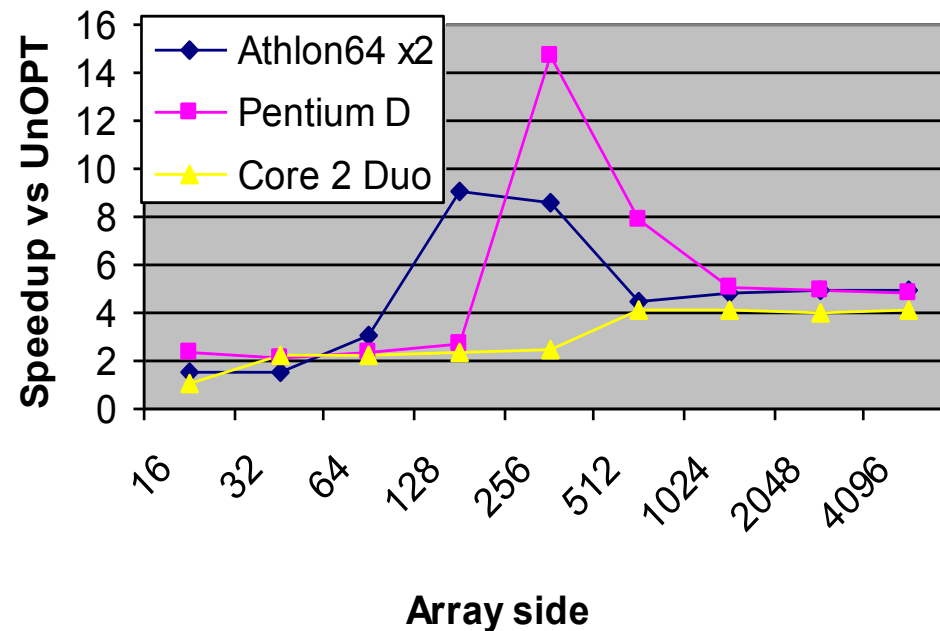
```
//Unoptimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A[i][j].d = A[i-1][j-1].d;
  }
}
```



```
struct vec_type
{
    char a;
    char b;
    char c;
    char d;
};
```

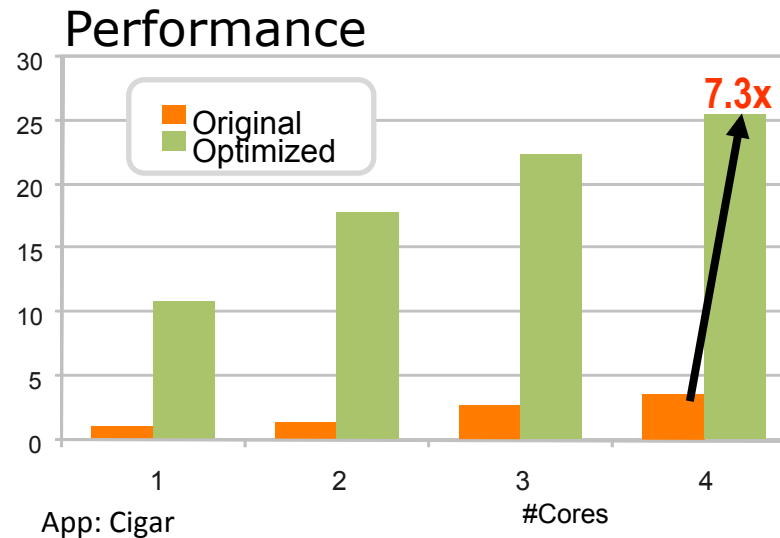


Performance Difference: Sparse Data





Example 2: Sparse data (Cigar)



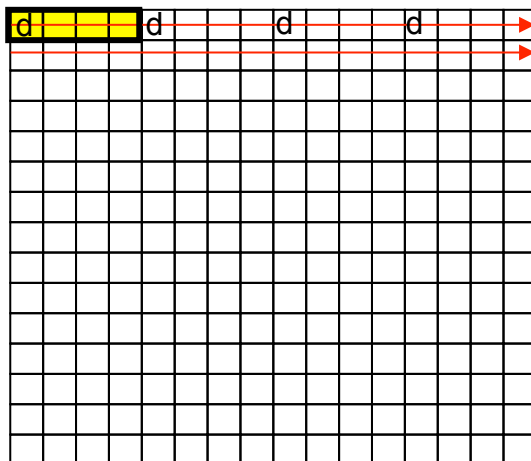
Original application scales linearly with # cores
... because slow execution is easy to scale

→ Duplicate one member of a struct into a single dense vector



Example: Sparse data allocation

```
sparse_rec sparse [HUGE];  
  
for (int j = 0; j < HUGE; j++)  
{  
    sparse[j].a = 'a'; sparse[j].b = 'b'; sparse[j].c = 'c'; sparse[j].d = 'd'; sparse[j].e = 'e';  
    sparse[j].f1 = 1.0; sparse[j].f2 = 1.0; sparse[j].f3 = 1.0; sparse[j].f4 = 1.0; sparse[j].f5 = 1.0;  
}
```



```
struct sparse_rec  
{  
    // size 80B  
    char a;  
    double f1;  
    char b;  
    double f2;  
    char c;  
    double f3;  
    char d;  
    double f4;  
    char e;  
    double f5;  
};
```

```
struct dense_rec  
{  
    //size 48B  
    double f1;  
    double f2;  
    double f3;  
    double f4;  
    double f5;  
    char a;  
    char b;  
    char c;  
    char d;  
    char e;  
};
```

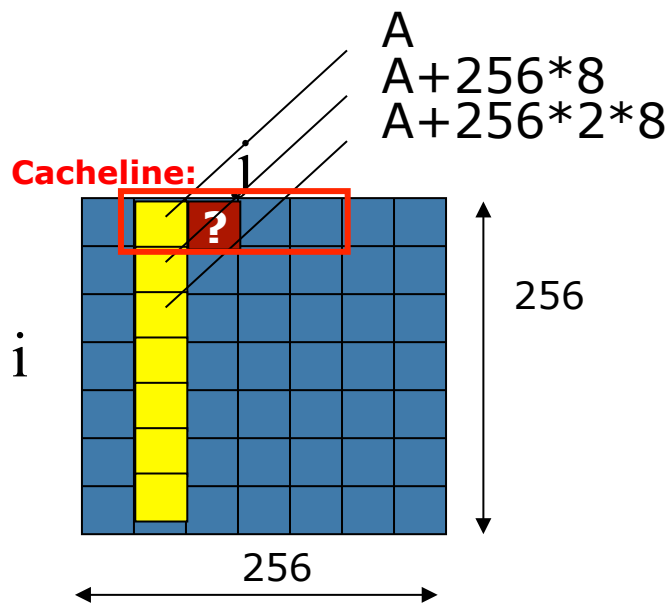


Loop Merging

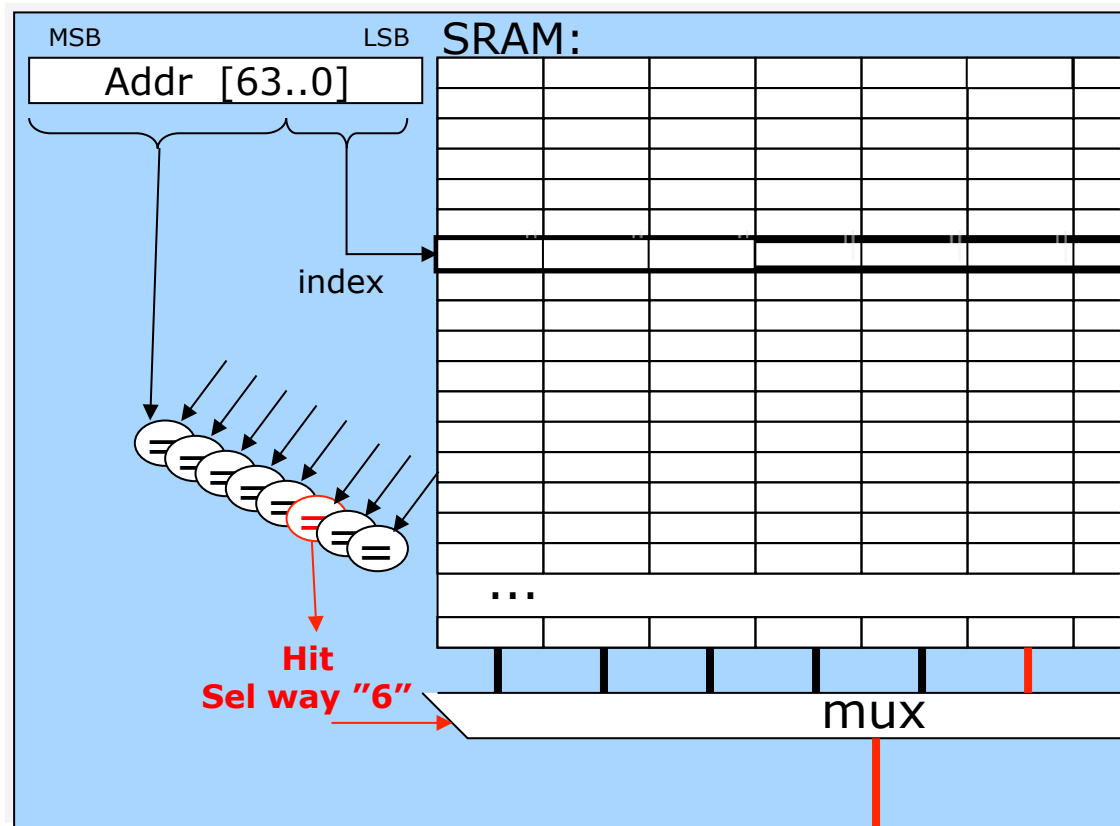
```
/* Unoptimized */
for (i = 0; i < N; i = i + 1)
    for (j = 0; j < N; j = j + 1)
        a[i][j] = 2 * b[i][j];
for (i = 0; i < N; i = i + 1)
    for (j = 0; j < N; j = j + 1)
        c[i][j] = K * b[i][j] + d[i][j]/2

/* Optimized */
for (i = 0; i < N; i = i + 1)
    for (j = 0; j < N; j = j + 1)
        a[i][j] = 2 * b[i][j];
        c[i][j] = K * b[i][j] + d[i][j]/2;
```

Padding of data structures

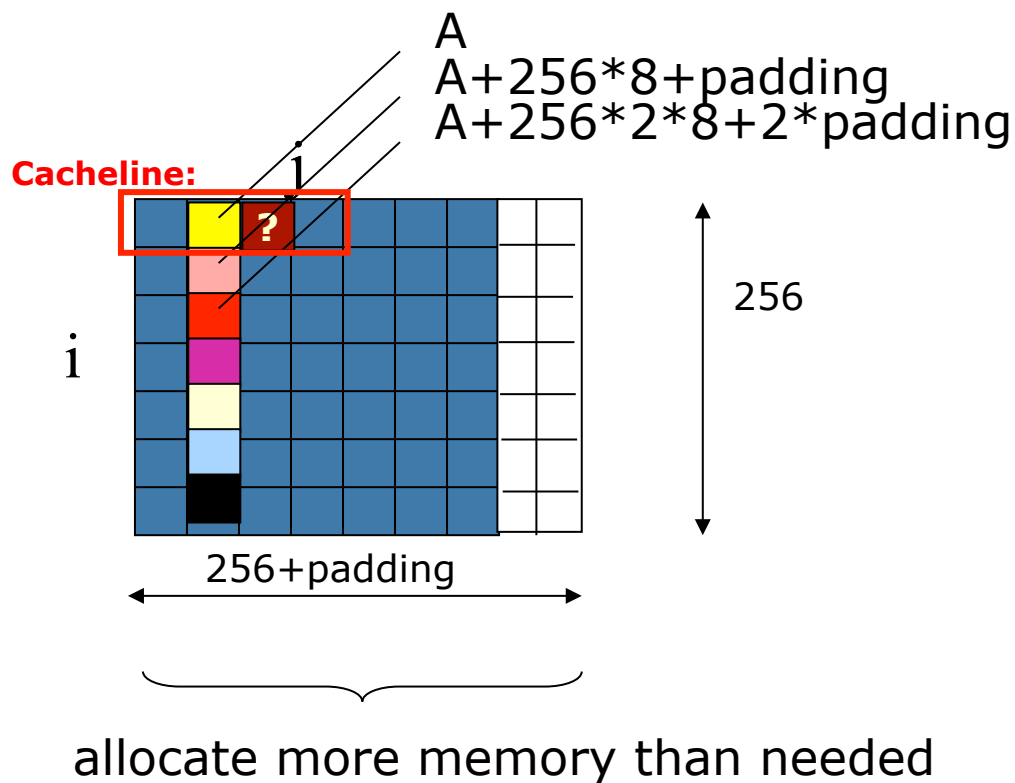


Generic Cache:

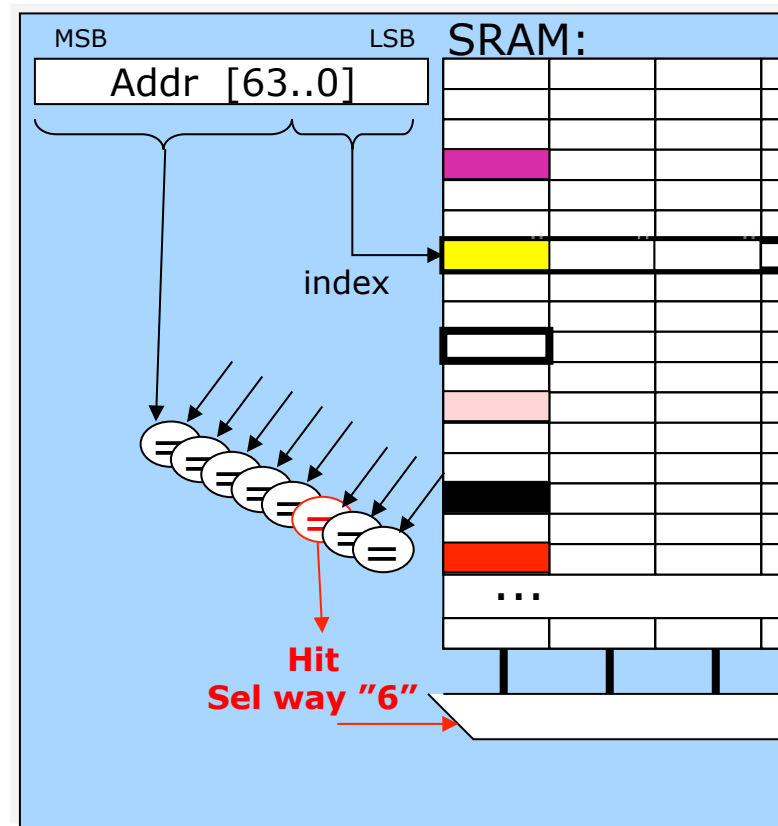


Data = 64B

Padding of data structures



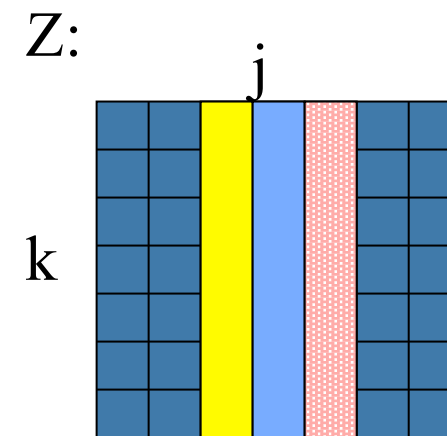
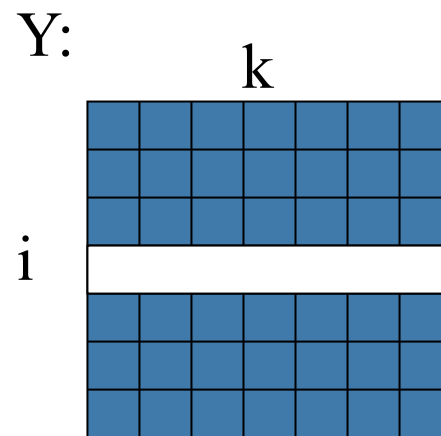
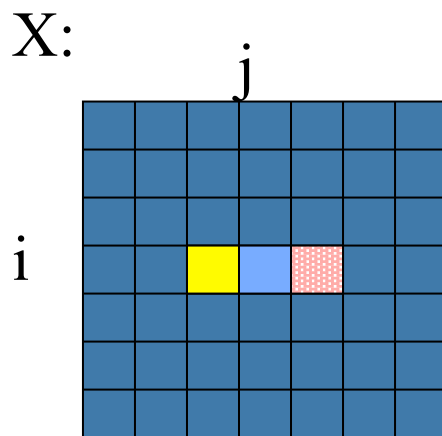
Generic Cache:





Blocking

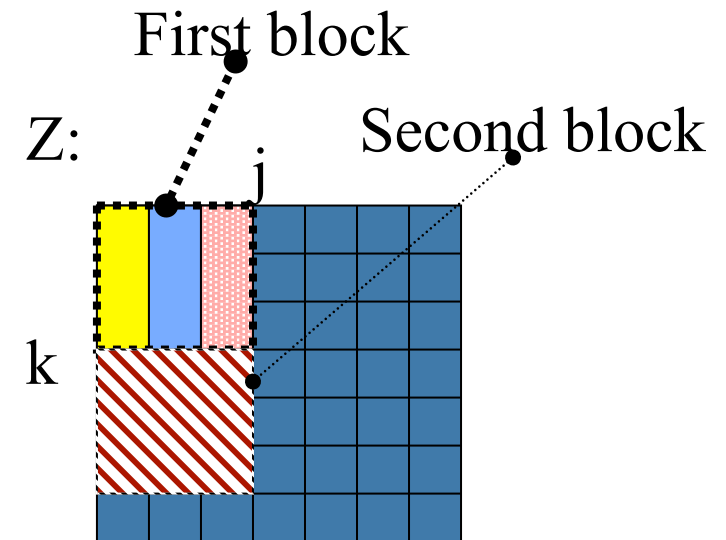
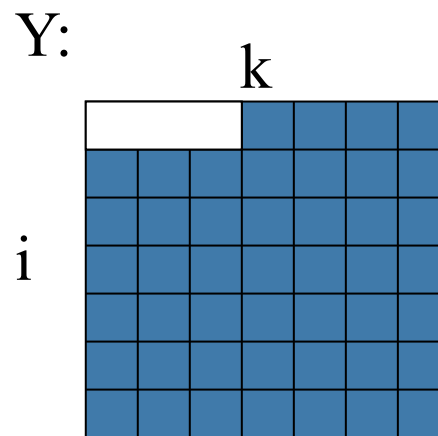
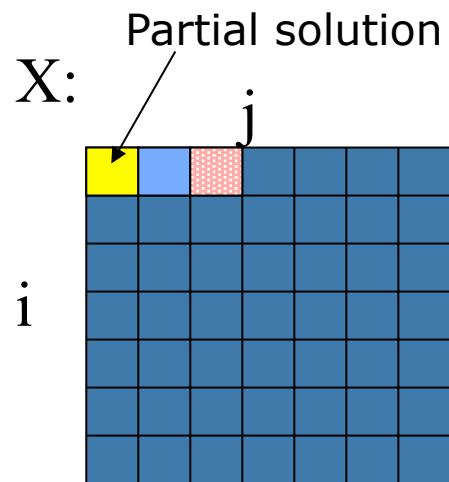
```
/* Unoptimized ARRAY: x = y * z */  
for (i = 0; i < N; i = i + 1)  
  for (j = 0; j < N; j = j + 1)  
    {r = 0;  
     for (k = 0; k < N; k = k + 1)  
       r = r + y[i][k] * z[k][j];  
     x[i][j] = r;  
    };
```





Blocking

```
/* Optimized ARRAY: X = Y * Z */  
for (jj = 0; jj < N; jj = jj + B)  
for (kk = 0; kk < N; kk = kk + B)  
for (i = 0; i < N; i = i + 1)  
  for (j = jj; j < min(jj+B,N); j = j + 1)  
    {r = 0;  
     for (k = kk; k < min(kk+B,N); k = k + 1)  
       r = r + y[i][k] * z[k][j];  
     x[i][j] += r;  
    };
```





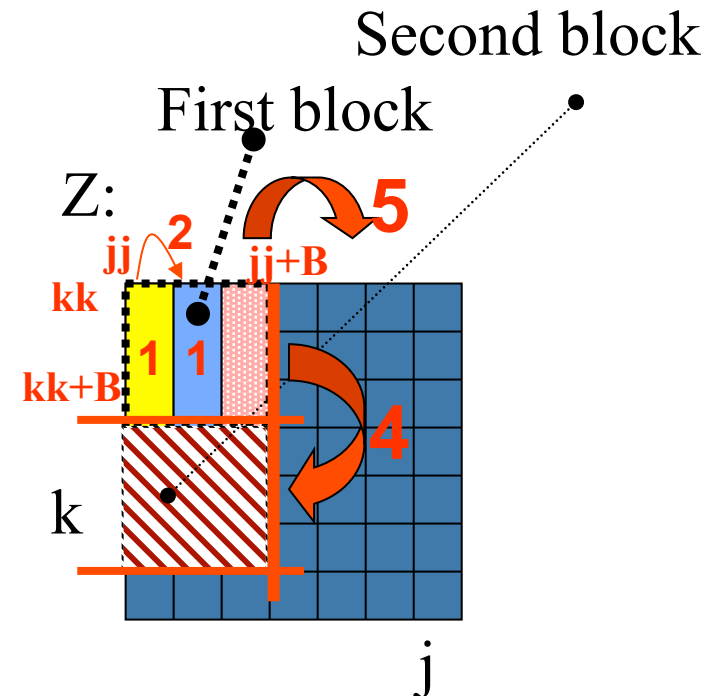
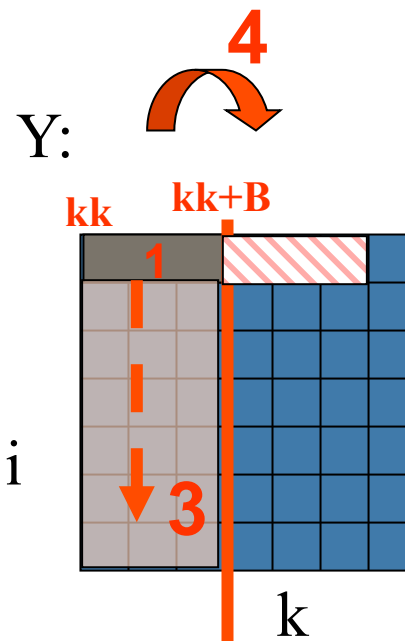
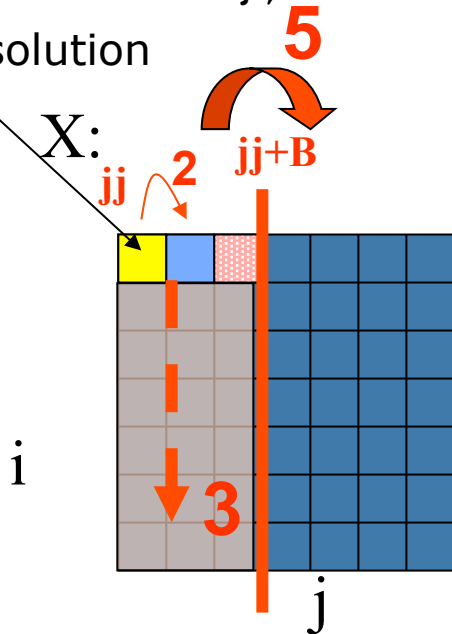
Blocking: the Movie!

```

/* Optimized ARRAY: X = Y * Z */
for (jj = 0; jj < N; jj = jj + B) /* Loop 5 */
for (kk = 0; kk < N; kk = kk + B) /* Loop 4 */
for (i = 0; i < N; i = i + 1) /* Loop 3 */
  for (j = jj; j < min(jj+B,N); j = j + 1) /* Loop 2 */
    {r = 0;
     for (k = kk; k < min(kk+B,N); k = k + 1) /* Loop 1 */
       r = r + y[i][k] * z[k][j];
     x[i][j] += r;
    };

```

Partial solution





Coherence traffic

ORIG:

Thread 0:

```
int a, total;
spawn_child()
for (int i; i < HUGE; i++) {
    /* do some work */
    a++;
}
join()
total = a;
```

Child:

```
for (int i; i < HUGE; i++) {
    /* do some work*/
    a++;
}
```

OPT:

Thread 0:

```
int a, total;
spawn_child()
for (int i; i < HUGE; i++) {
    /* do some work */
    a++;
}
join()
total += a;
```

Child:

```
int b;
for (int i; i < HUGE; i++) {
    /* do some work */
    b++;
}
total += b;
```



False sharing

ORIG:

Thread 0:

```
int a, b;  
spawn_child()  
for (int i; i < HUGE; i++) {  
    ...  
    a++;  
}  
join()  
total = a + b;
```

Child:

```
for (int i; i < HUGE; i++) {  
    ...  
    b++;  
}
```

OPT:

Thread 0:

```
int a;  
spawn_child()  
for (int i; i < HUGE; i++) {  
    ...  
    a++;  
}  
join()  
total += a;
```

Child:

```
int b;  
for (int i; i < HUGE; i++) {  
    ...  
    total += b;  
}
```



Coherence Utilization

ORIG:

```

struct vec_type
{
    int a;
    int b;
    int c;
    int d;
    int e;
    int f;
};

```

x[0]	x[12]	x[
abcde f	abcde f	ab

Thread 0:

```

vec_type x[HUGE];
for (int i; i < HUGE; i++) {
    ...
    x[i].a++;
}
spawn_child()
...
join()

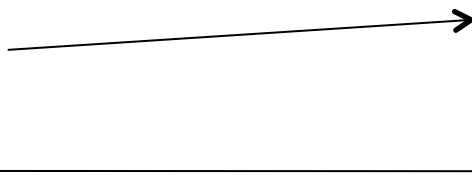
```

Child (Thread 1)

```

for (int i; i < HUGE; i++) {
    y[i] = x[i].a;
}

```





UPMARC

Uppsala Programming for
Multicore Architectures
Research Center

- Uppsala Programming for Multicore Architecture Center
- 62 MSEK grant / 10 years [\$9M/10y]
+ related additional grants at UU = 130MSEK
- Research areas:

- Erik:**
- ✱ Performance modeling
 - ✱ New parallel algorithms
 - ✱ Scheduling of threads and resources
 - ✱ Testing & verification
 - ✱ Language technology
 - ✱ MC in wireless and sensors



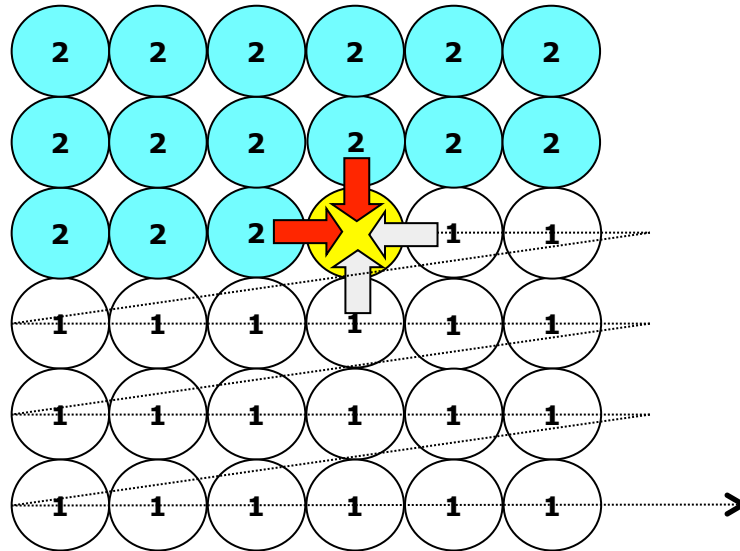
Multi-threaded Case Study: Gauss-Seidel on Multicores

From Wallin et al, ICS 2006



Example: Gauss Seidel

Mission: “Maximize the parallelism and minimize the inter-thread communication”

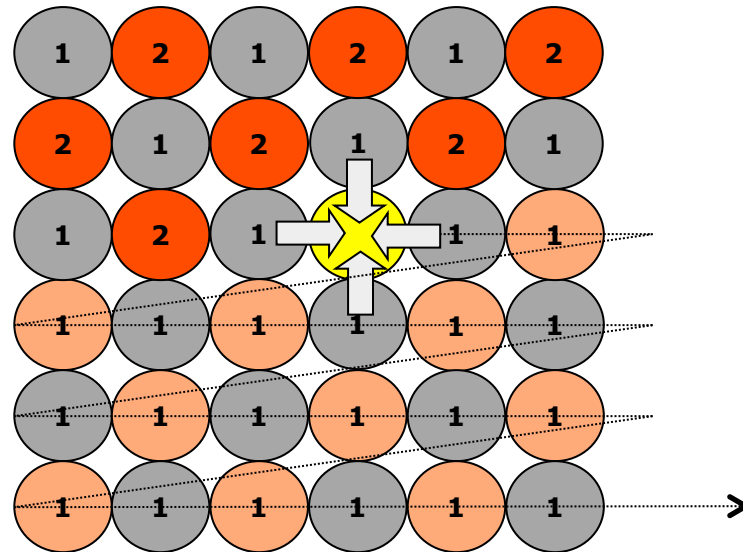


LOOP:
UPDATE ALL POINTS
IF (convergence_test)
<done>

(Longer explanation: [Finding a Door in the Memory Wall @ HPCWire](#))

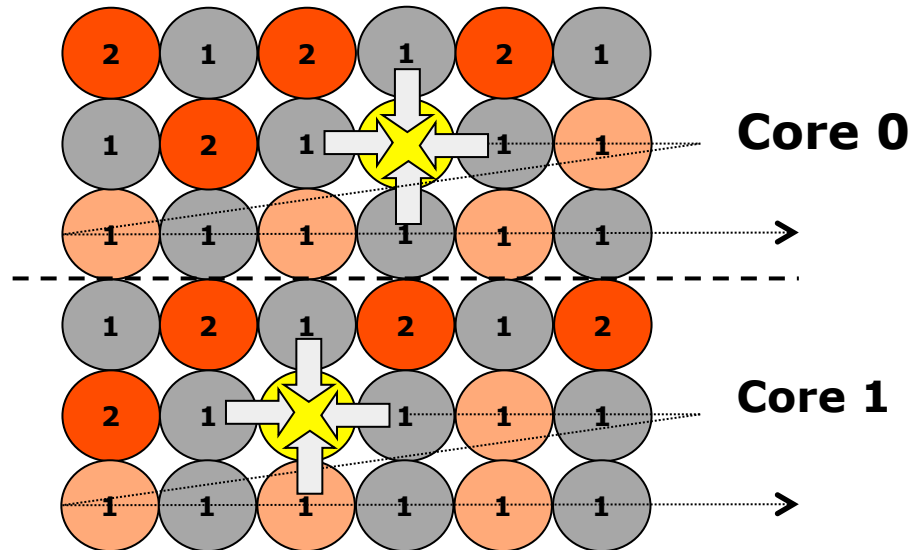
SW Optimizations 25

State-of-the-art: Removing Dependence: Red/Black



```
LOOP:  
  UPDATE ALL RED POINTS  
  UPDATE ALL BLACK POINTS  
  IF (convergence_test)  
    <done>
```

State-of-the-art: Red/Black, Parallelism = $N^2/2$



LOOP:

IN PARALLEL: UPDATE ALL **RED** POINTS

<barrier>

IN PARALELL: UPDATE ALL **BLACK** POINTS

<barrier>

IF (convergence_test)

<done>

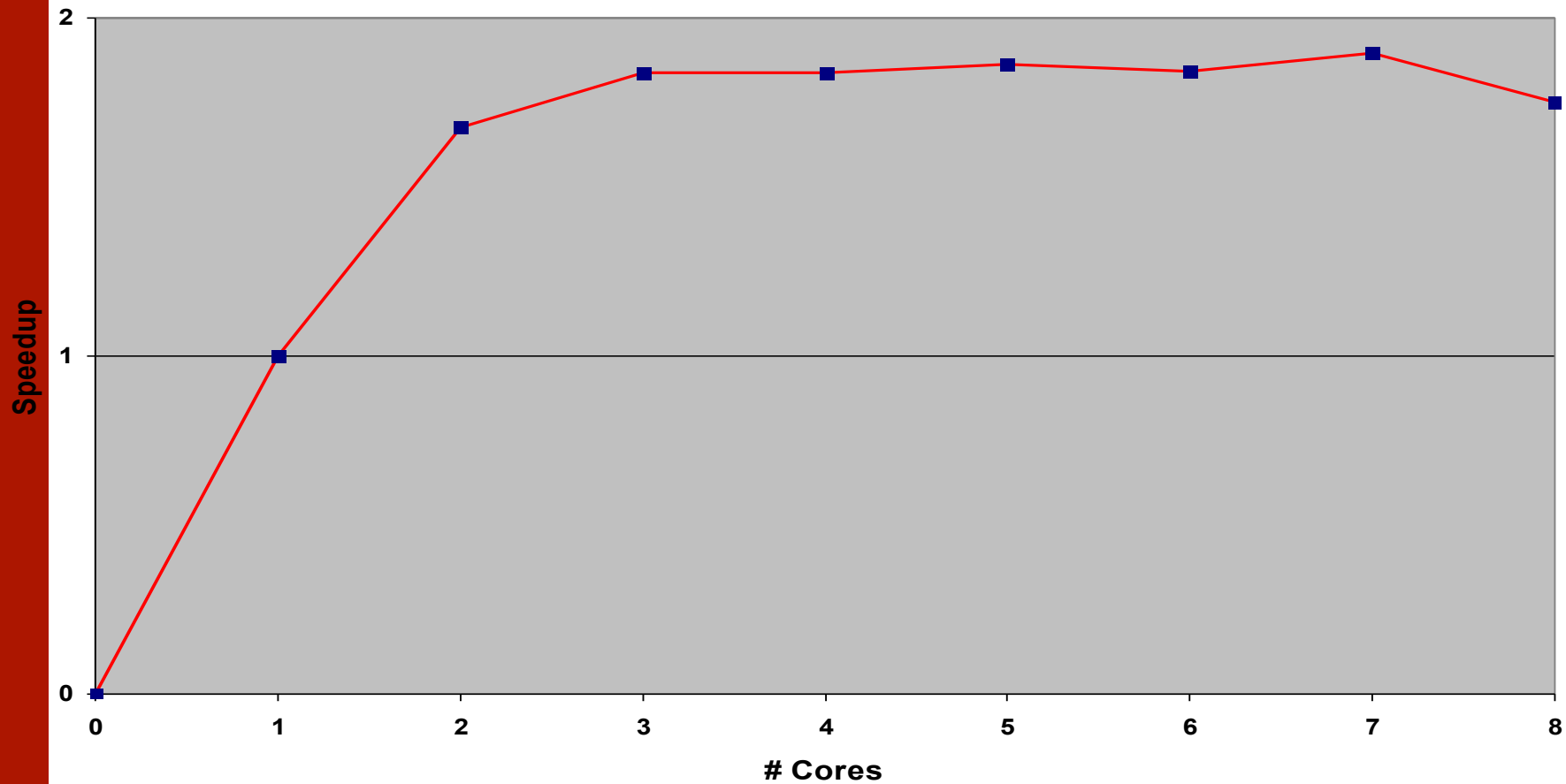
Limited communication ☺

$N^2/2$ parallelism ☺

Done!

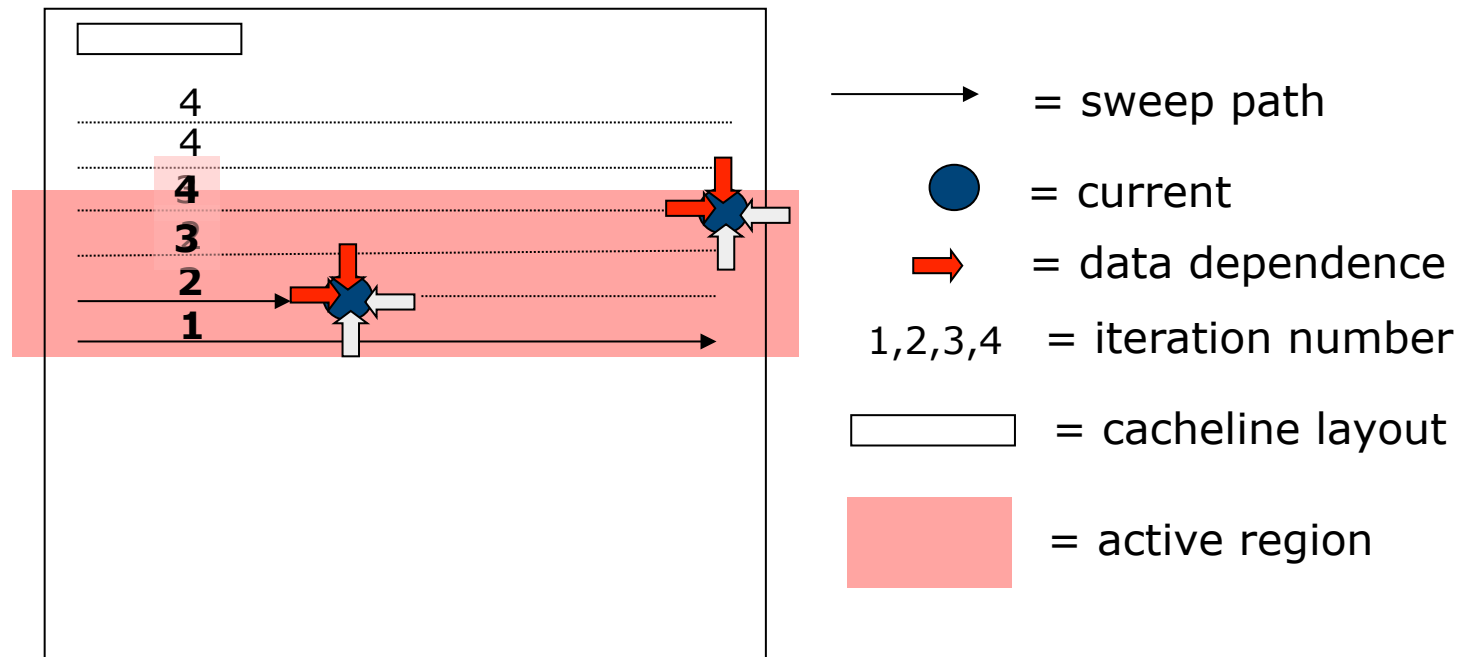
Only one problem...

Only One Problem: Performance



Back to the drawing board: Temporal blocking for seq. code

Communication is "for free" and moderate parallelism is OK
Priority 1: limit bandwidth needs!



LOOP:

 LOOP:

 UPDATE ALL POINTS IN ACTIVE REGION

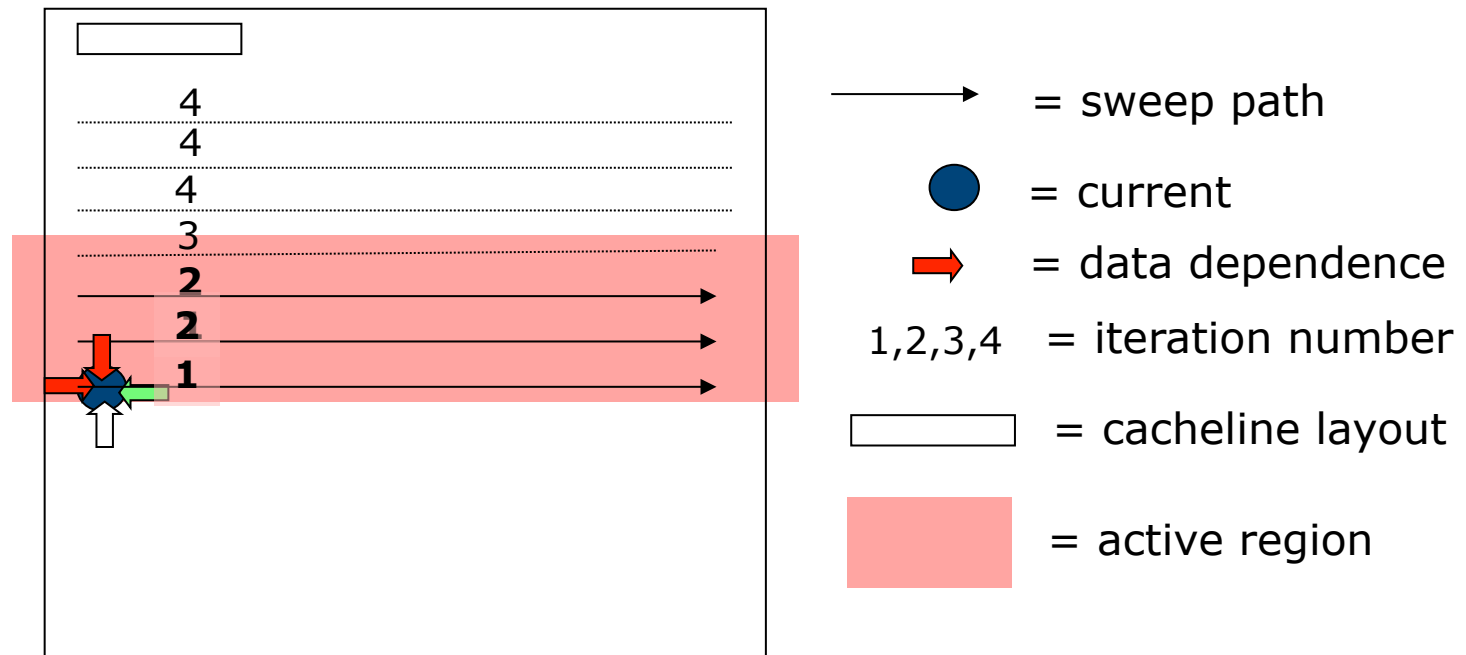
 SLIDE DOWN THE REGION

 IF (convergence_test)

 <done>

Back to the drawing board: Temporal blocking for seq. code

Communication is "for free" and moderate parallelism is OK
Priority 1: limit bandwidth need!



LOOP:

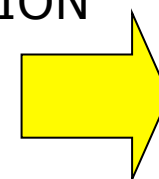
 LOOP:

 UPDATE ALL POINTS IN ACTIVE REGION

 SLIDE DOWN THE REGION

 IF (convergence_test)

 <done>

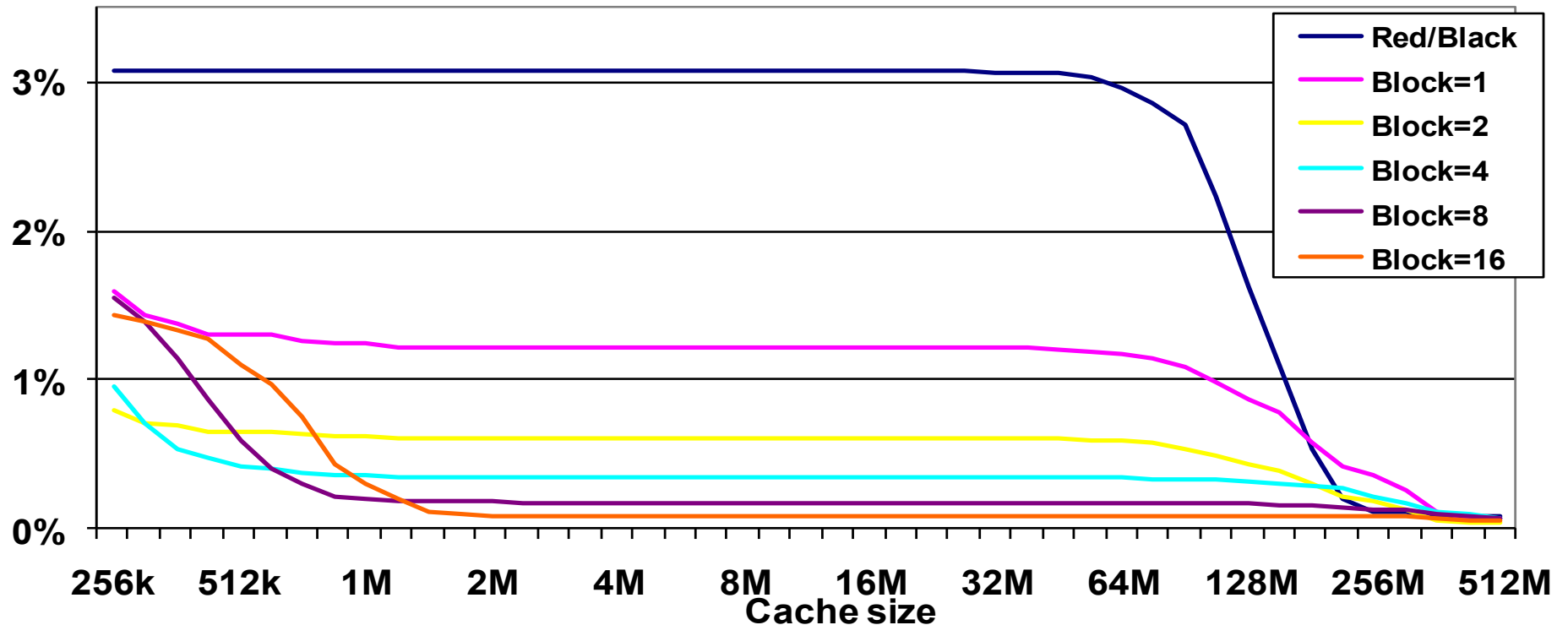


**4 iterations in
one sweep!**



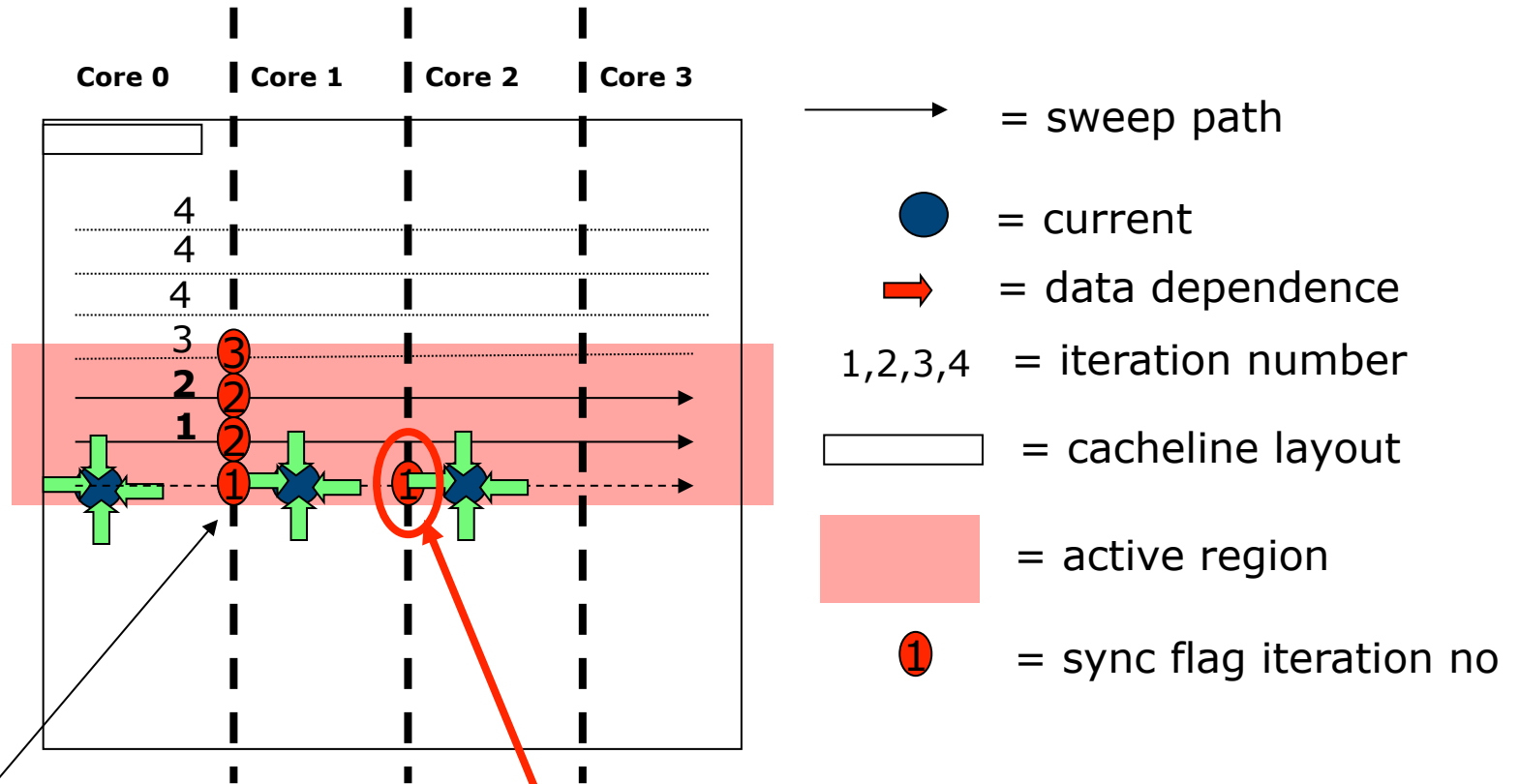
DRAM_traffic(cache_size)

Fetch Rate,
i.e, fraction of mem_ops generating DRAM traffic





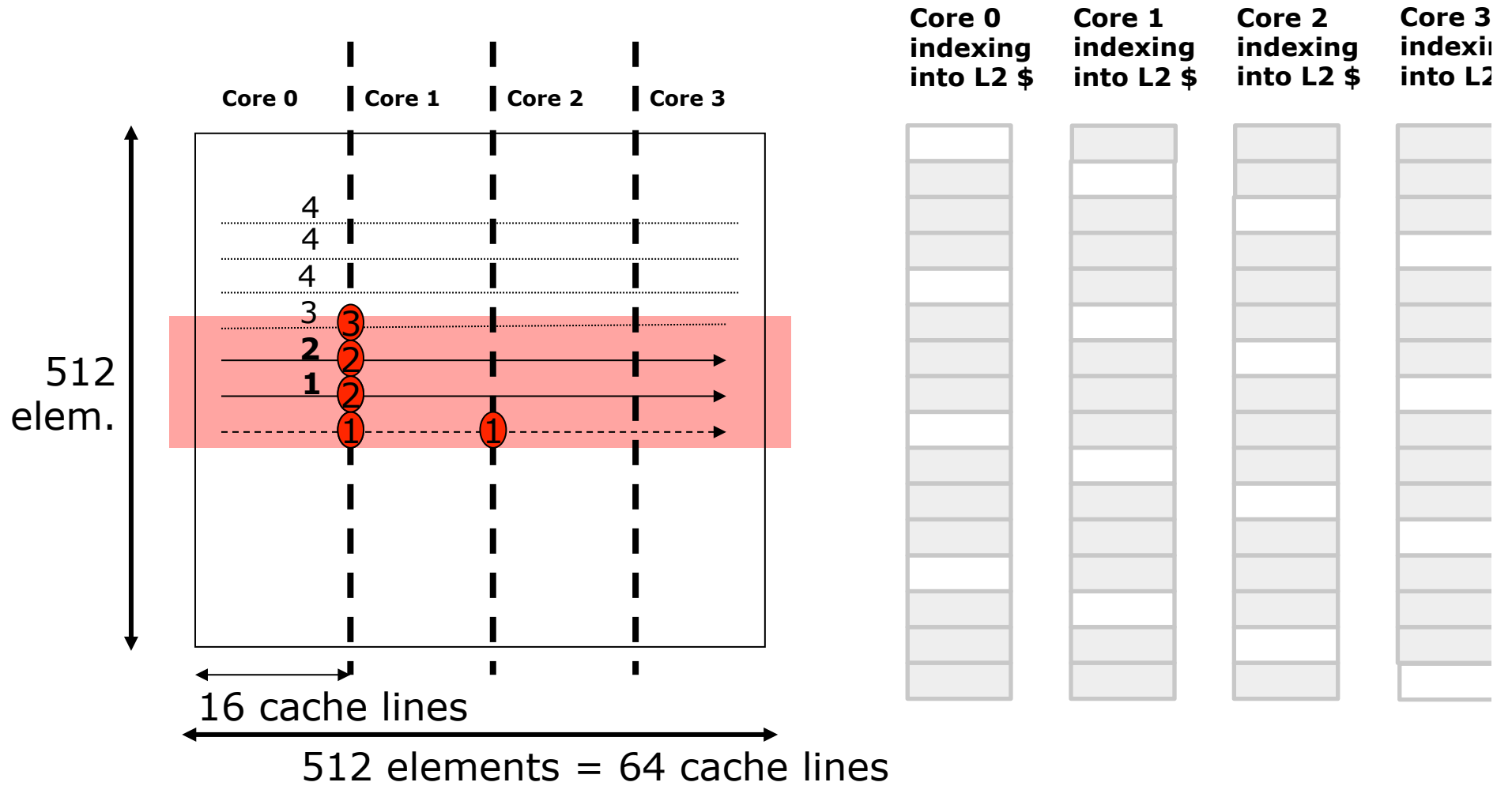
G-S, temp block Parallelism = N



Synchronization
flags

Wait until "lefty" is done:
Lots of communication ☹️
 • Producer/Consumer Flag
 • Sharing of data values
Only N-fold parallelism ☹️

Problems we ran into 1 (2)



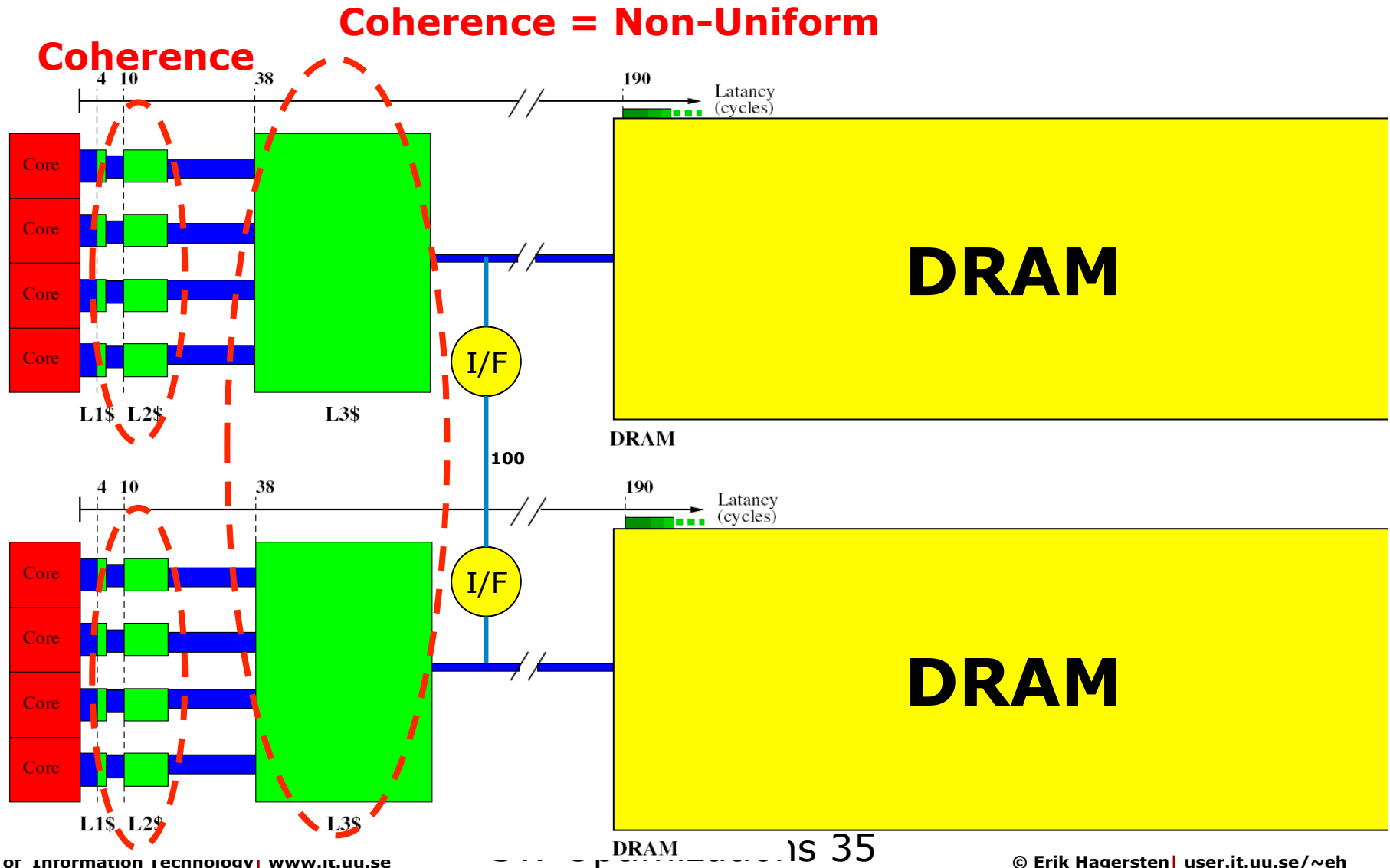


Problems we ran into 2 (2)

- We had a loop nesting problem that the compiler optimized away
- ... sometimes

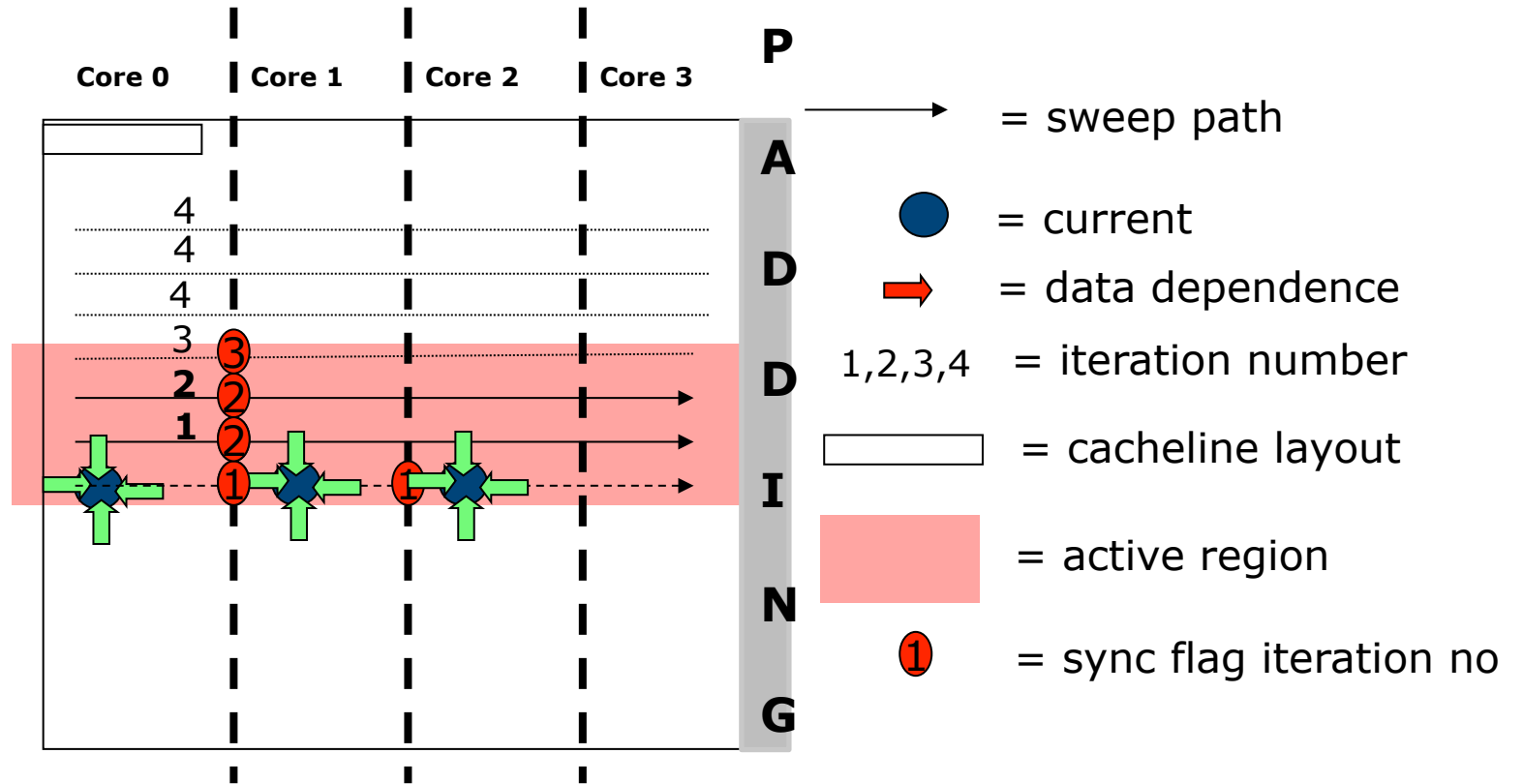


Running on a Multisocket



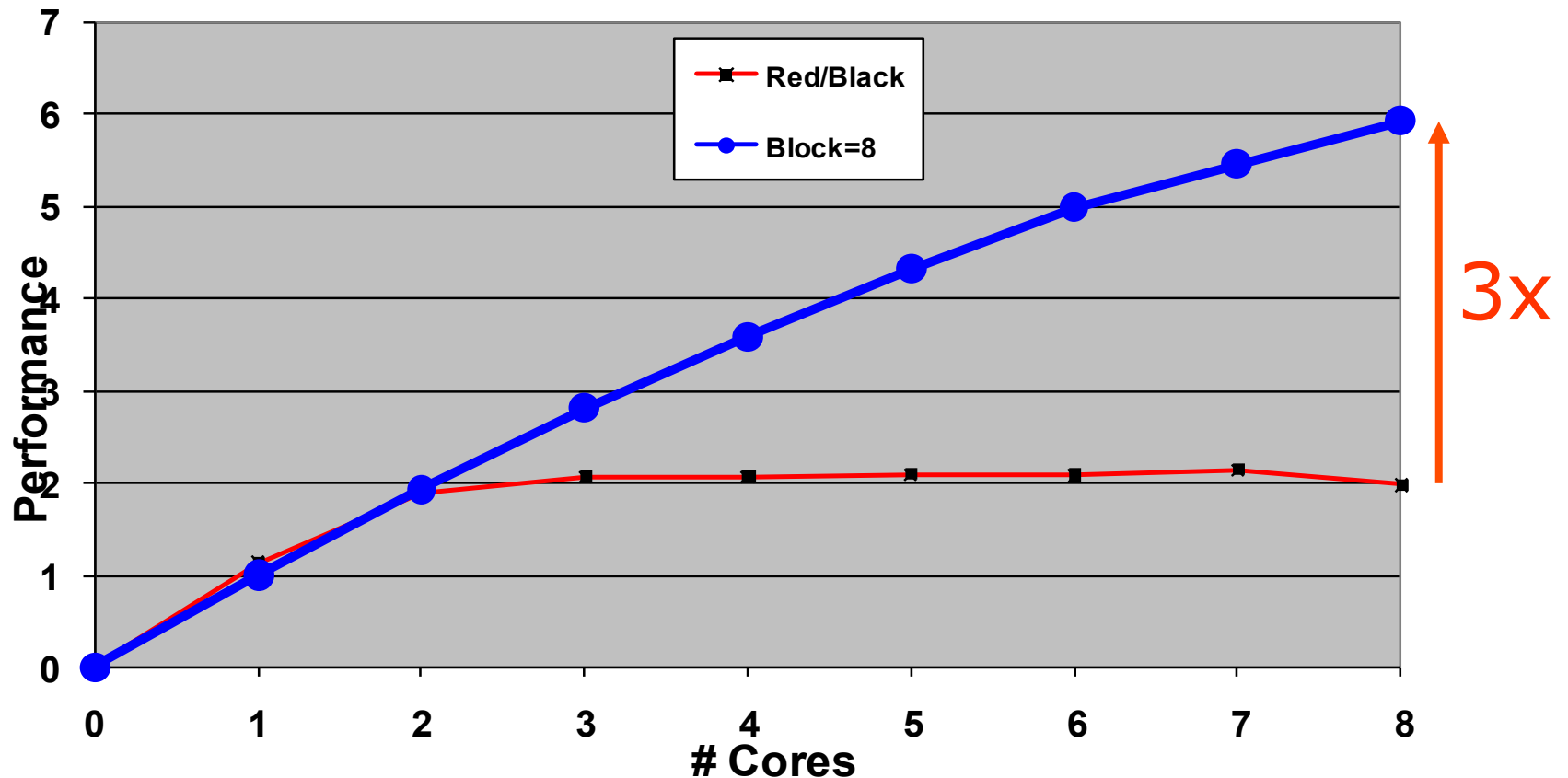


Example: G-S, temp blocking





Lessons Learned: Optimize cache usage **BEFORE** parallelizing



[Wallin, Löf, Holmgren, Hagersten @ ICS 2006]

SW Optimizations 37