Programming MPs

Different Programming Paradigms

Inter-Process Communication (IPC)

Processes can be made to share memory (e.g. mmap())

Process:
Process ID
Virtual2Physical mapping
Process-global storage (Shared mem)
File handlers
Executable code…

Thread:
ThreadID
Thread-local storage
Execution context (Regs)
Stack

Process:

Thread:

MPI library

- Supports **process creation**
- Supports **data movements between processes**
  - Send_message, Receive_message,...
  - One-sided (non-coherent) communication
- Synchronization
- Collective communication
  - Scatter/gather, Reduction, Barriers...

UPPSALA UNIVERSITET

Pthreads library
OpenMP compilers

- Supports **thread creation**
- Supports **shared memory between threads**
- Synchronization
- Collective communication
  - Reduction, Barriers...

- Each MPI process may contain several threads (hybrid programming)

AVDARK 2013

Pthreads Example

```
#include <pthread.h> ...
#define NUM_THREADS    5

void *TaskCode(void *argument)
{  int tid;
   tid = *((int *) argument);
   printf("Hello World! It's me, thread %d!\n", tid); /* more useful stuff here */
   return NULL; }

int main(void)
{  pthread_t threads[NUM_THREADS];
   int thr_args[NUM_THREADS];
   int rc, i;

   for (i=0; i<NUM_THREADS; ++i) { /* create all threads */
     thr_args[i] = i;
     printf("In main: creating thread %d\n", i);
     rc = pthread_create(&threads[i], NULL, TaskCode, (void *) &thr_args[i])
     assert(0 == rc);}


   for (i=0; i<NUM_THREADS; ++i) { /* wait for all threads to complete */
     rc = pthread_join(&threads[i], NULL);
     assert(0 == rc); }

   exit(EXIT_SUCCESS);
}
```

OpenMP example

UPPSALA UNIVERSITET

```
#define NUM_THREADS    5

int main(void)
{  int  i;

   #pragma omp parallel for, local(i)
   for (i=0; i<NUM_THREADS; ++i) {  /* create all threads */
     printf("Hello World! It's me, thread %d!\n", i)
  /* implicit barrier here, waiting for all threads to complete */
}
```

UPPSALA UNIVERSITET

Open MP in a nutshell

```
//Matrix multiplication using OpenMP:

void M_mult(float A[n][n], float B[n][n], float C[n][n]) {
  int i,j,k;
  float sum;
  #pragma omp parallel for, local(i,j,k,sum)
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      sum = 0;
      for (k=0; k<n; k++)
        sum=sum+A[i][k]*B[k][j];
      C[i][j]=sum;
    }
  //Here is an implicit barrier implemented by openMP
  }
}
```

Specifying which variables that should be thread-local

Note: This code can still be compiled with an ordinary C compiler

AVDARK 2013

Examples of OpenMP pragmas

#pragma omp for
The loop construct specifies that the iterations of loops will be distributed among and executed by the encountering team of threads.

#pragma omp parallel
The parallel construct forms a team of threads and starts parallel execution.

#pragma omp critical
The critical construct restricts execution of the associated structured block to a single thread at a time.

#pragma omp barrier
The barrier construct specifies an explicit barrier at the point at which the construct appears.

#pragma omp flush
The flush construct executes the OpenMP flush operation, which makes a thread's temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables. **(~memory fence)**

...

Environment variable examples:

OMP_SCHEDULE type [,chunk]
Sets the run-sched-var for the runtime schedule type and chunk size. Valid OpenMP schedule types are static, dynamic, guided, or auto. Chunk is a positive integer.

OMP_NUM_THREADS num
Sets the nthreads-var for the number of threads to use for Parallel regions

...

Runtime library examples:

int omp_get_max_threads(void);
Returns maximum number of threads that could be used to form a new team using a "parallel" construct without a "num_threads" clause.

void omp_set_num_threads(int num_threads);
Affects the number of threads used for subsequent parallel regions that do not specify a num_threads clause.

int omp_get_num_threads(void);
Returns the number of threads in the current team.

int omp_get_thread_num(void);
Returns the ID of the encountering thread where ID ranges from zero to the size of the team minus 1.

...

UPPSALA UNIVERSITET

What is wrong

What should be fixed?

- Loop i has data dependencies and cannot be parallelized
- “sum” should also be local
- You have to add #pragma omp barrier at the end

```
//Matrix multiplication using OpenMP:

void M_mult(float A[n][n], float B[n][n], float C[n][n]) {
  int i,j,k;
  float sum;
  #pragma omp parallel for, local(i,j,k)
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      sum = 0;
      for (k=0; k<n; k++)
        sum=sum+A[i][k]*B[k][j];
      C[i][j]=sum;
    }
  }
}
```

UPPSALA UNIVERSITET

How many threads?

```
//Matrix multiplication using OpenMP:

void M_mult(float A[n][n], float B[n][n], float C[n][n]) {
  int i,j,k;
  float sum;
  #pragma omp parallel for, local(i,j,k,sum)
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      sum = 0;
      for (k=0; k<n; k++)
        sum=sum+A[i][k]*B[k][j];
      C[i][j]=sum;
    }
  }
}
```

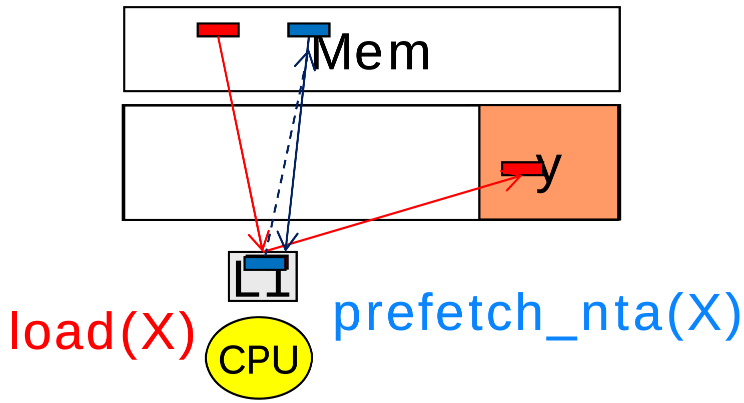
How many threads will be used?

- ❑ Always the same as total number of cores in the system
- ❑ One thread
- ❑ Five threads
- ❑ That is decided by the omp envirnoment vairables

Optimizing for MP:s

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

Cache Waste



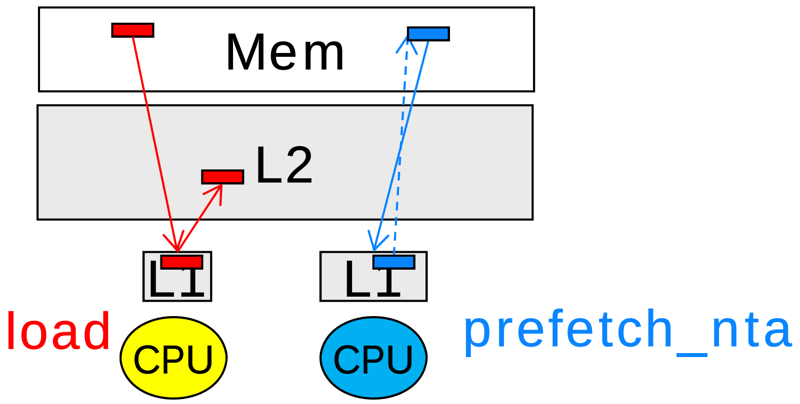

```
/* Unoptimized */
for (s = 0; s < ITERATIONS; s++){
   for (j = 0; j < HUGE; j++)
        x[j] = x[j+1];          /* will hog the cache but not benefit*/
   for (i = 0; i < SMALLER_THAN_L2_CACHE; i++)
        y[i] = y[i+1];          /* will be evicted between usages /*
}
/* Optimized */
for (s = 0; s < ITERATIONS; s++){
   for (j = 0; j < HUGE; j++) {
        PREFETCH_NTA x[j+1] /* will be installed
         x[j] = x[j+1];
   for (i = 0; I < SMALLER_THAN_L2_CACHE; i++)
        y[i] = y[i+1];      /* will always hit in the cache*/
}
```

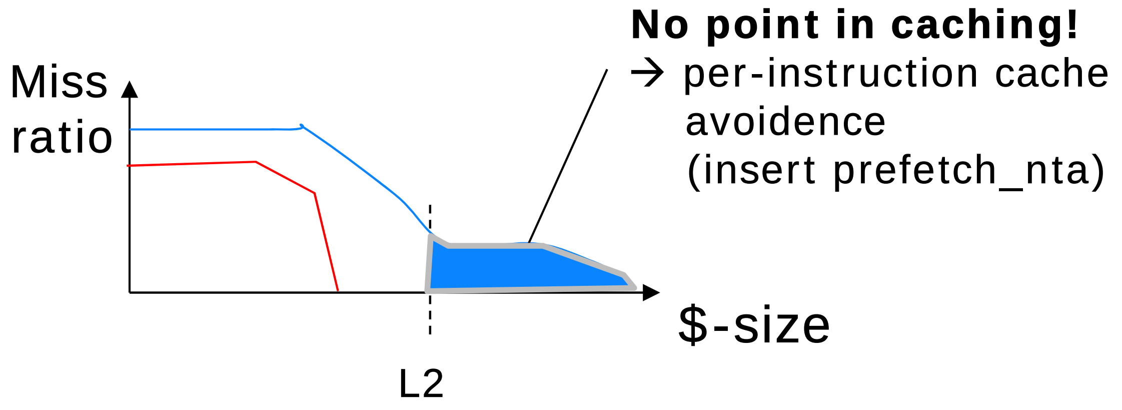
What kind of misses are removed?

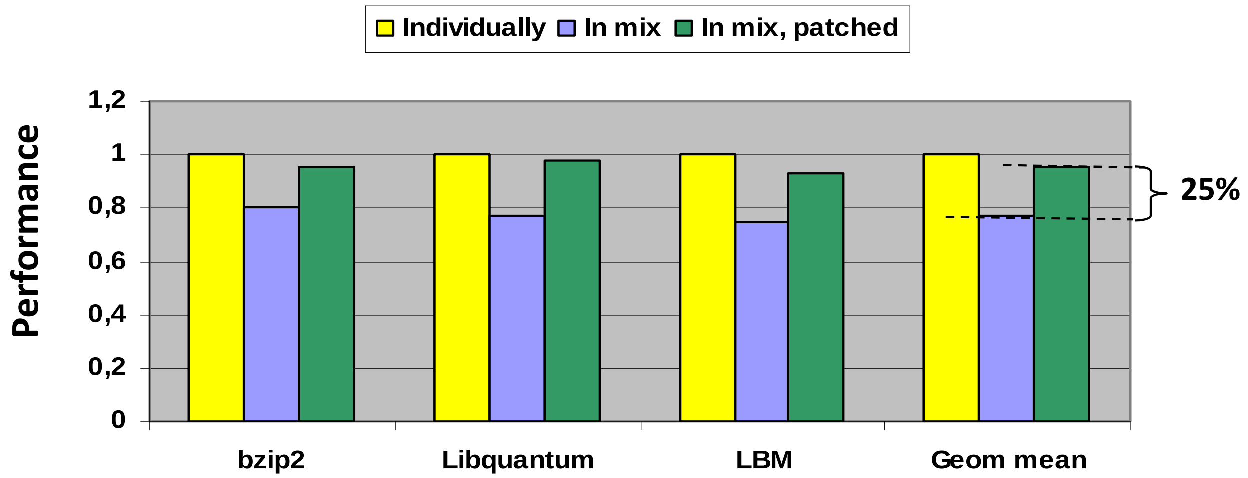
- ❑ Capacity
- ❑ Conflict
- ❑ Compulsory
- ❑ Coherence

➔ Can also be beneficial if applications are co-scheduled on MC and share cache with other applications.

UART: Avoiding cache waste (A. Sandberg)







AMD Opteron

Avoiding coherence traffic

What kind of misses are removed?
- ❑ Capacity
- ❑ Conflict
- ❑ Compulsory
- ❑ Coherence

ORIG:

Thread 0:

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

Child:

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

OPT:

Thread 0:

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

Child:

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

Avoiding false sharing

What can be fixed?

- ❑ Change the Child loop ctr to j
- ❑ Change the declaration of b
- ❑ Move ”join” last in Thread 0

ORIG:

Thread 0:

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

Child:

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

OPT:

Thread 0:

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

Child:

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


UPPSALA UNIVERSITET

AVDARK 2013

Coherence Utilization

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

x[0]						x[1]				
a	b	c	d	e	f	a	b	c	d	e

ORIG:

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;
 }
end_child()
```

OPT:

Thread 0:

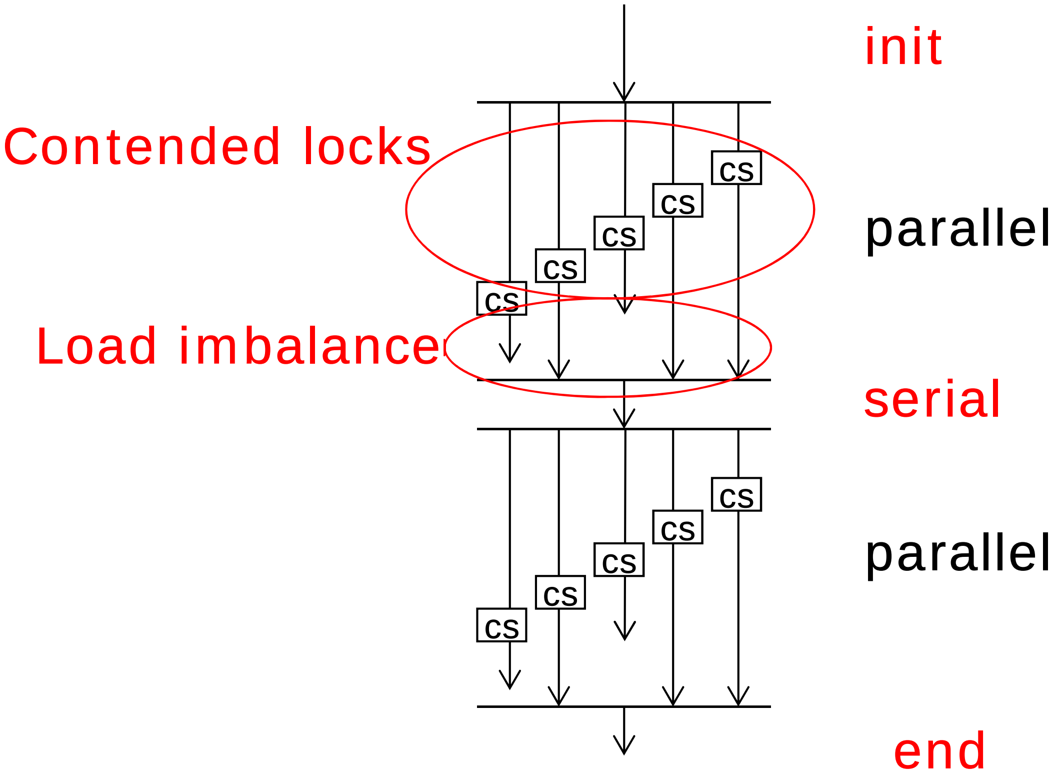
```
int x_a[HUGE];
for (int i; i< HUGE; i++) {
   ...
    x_a[i]++;
 }
spawn_child()
...
join()
```

Child (Thread 1)

```
for (int i; i< HUGE; i++) {
    y[i] = x_a[i];
 }
end_child()
```

Limits to Scalability

How much parallelism?

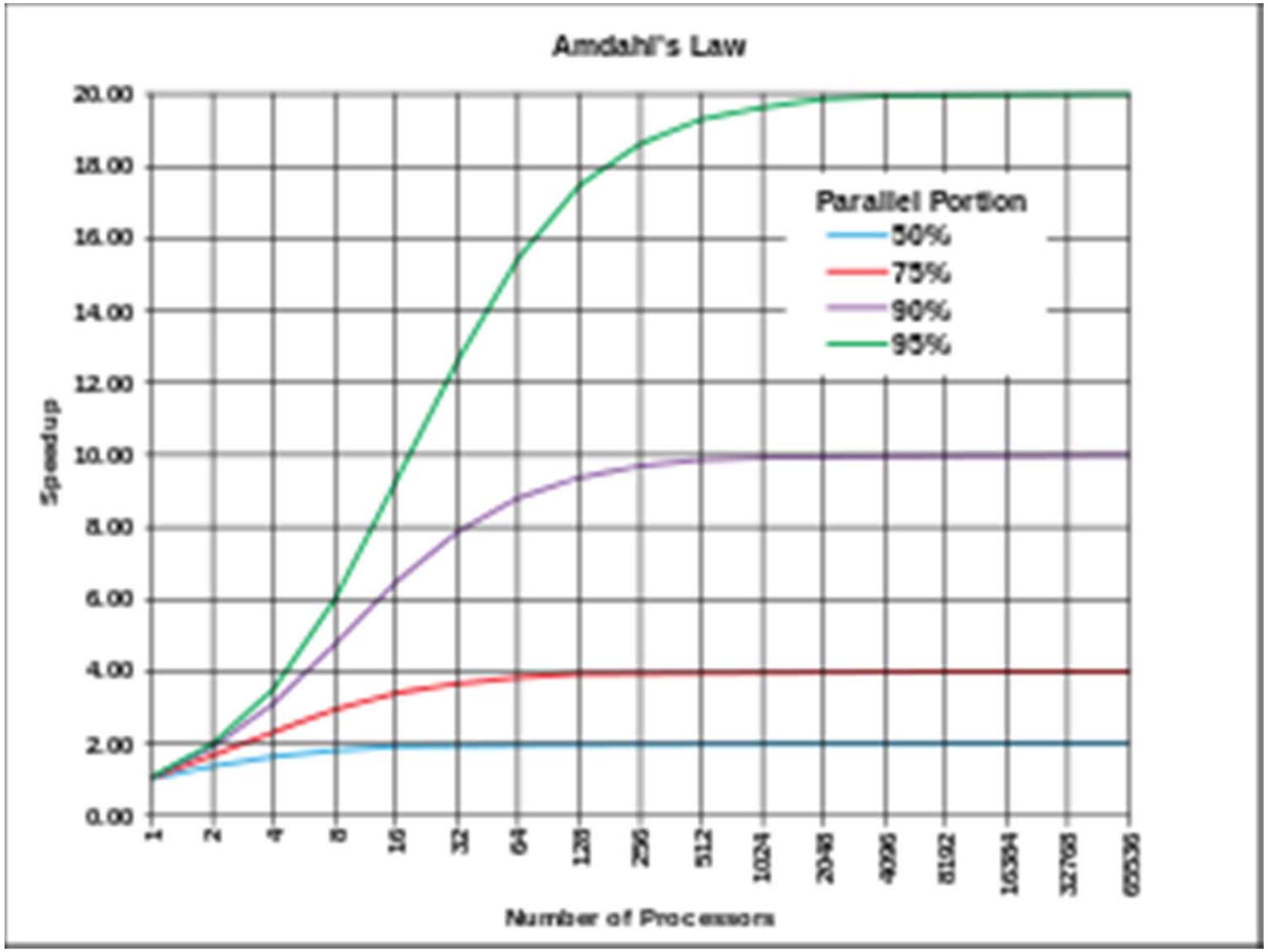


Amdahl's Law

- What fraction of the execution is serial?
  - Sequential segments
  - Contended critical sections
  - Waiting at a barrier (e.g. load imbalance)
  - Time to create/end a thread/process
  - Data initialization and I/O
  - Time to open N files
  - Parallel region limited by system data bandwidth...

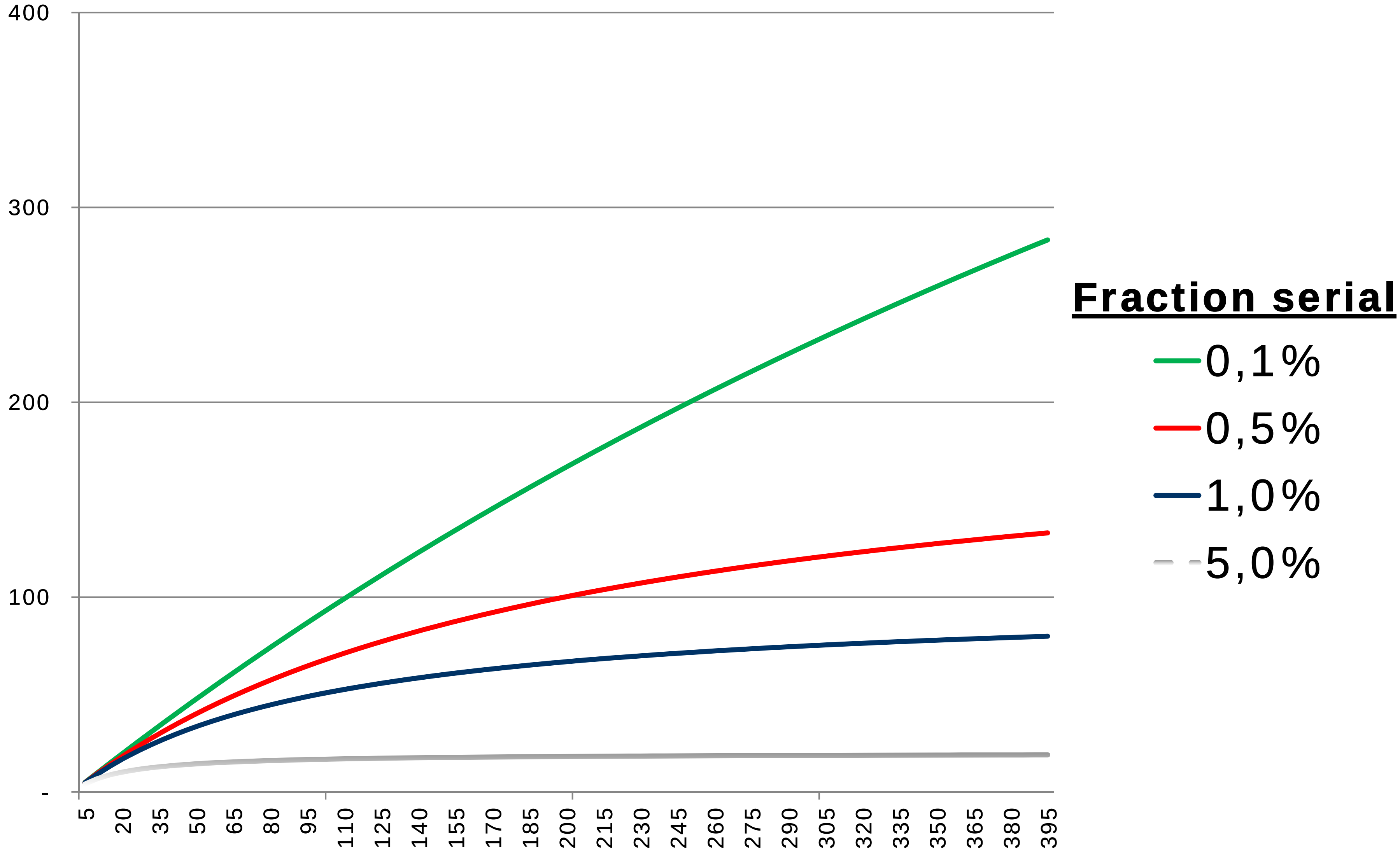
$$S(P) = 1 / [\alpha - 1/P(1- \alpha)]$$

where *P* is the number of processors, *S* is the speedup, and α the non-parallelizable fraction of any parallel process



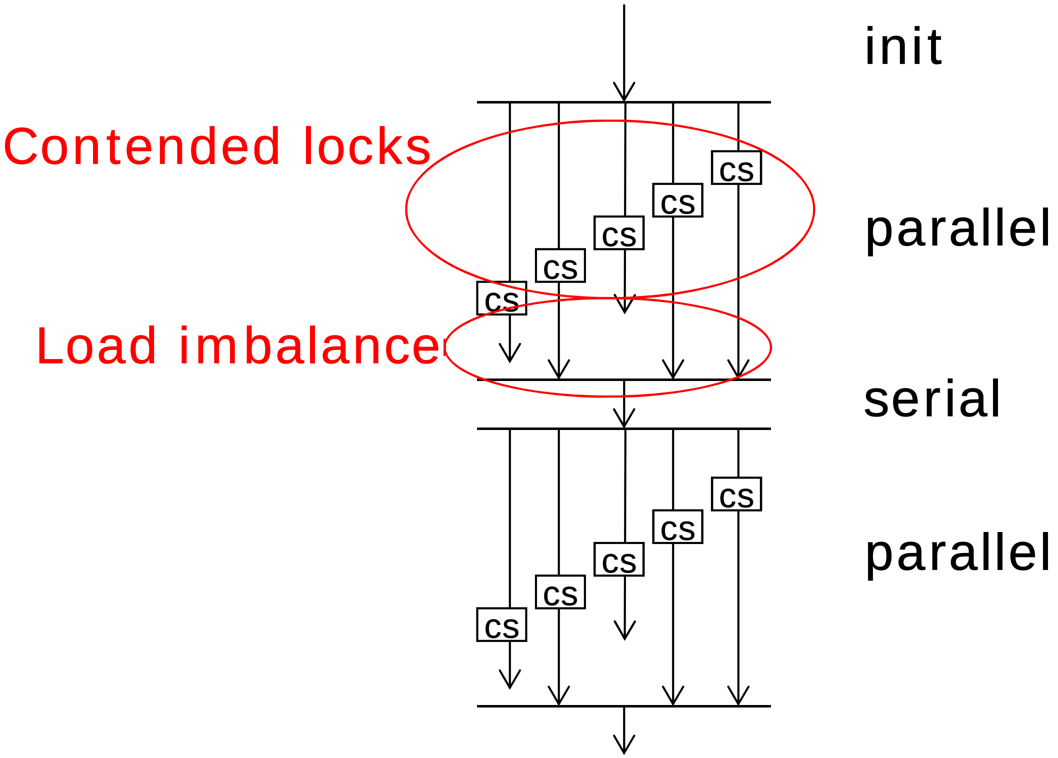
Amdahl's Law
Parallel Portion
50%
75%
90%
95%
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

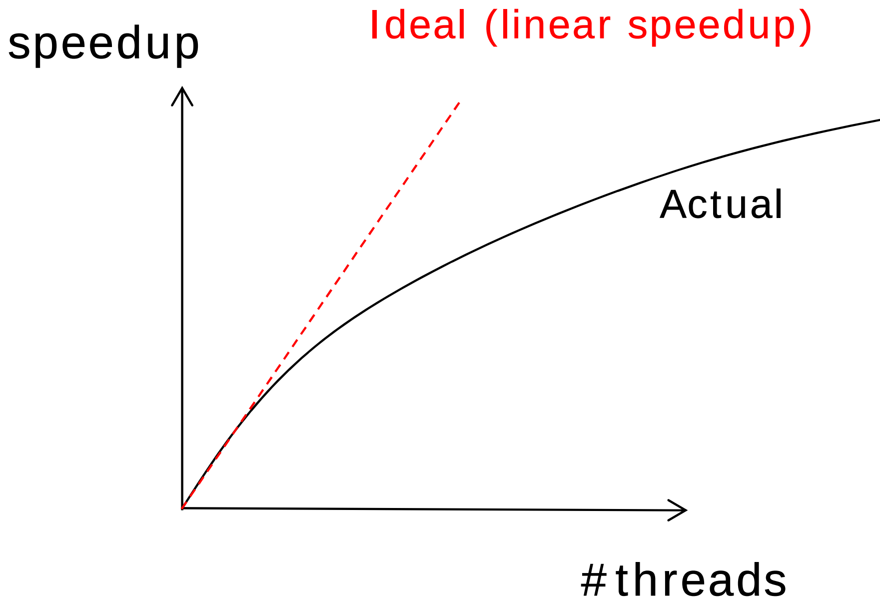
Speedup


400
300
200
100
-
5
20
35
50
65
80
95
110
125
140
155
170
185
200
215
230
245
260
275
290
305
320
335
350
365
380
395
Fraction serial
0,1%
0,5%
1,0%
5,0%
Processors

UPPSALA UNIVERSITET

How much parallelism?





AVDARK 2013

Gustavsson’s Law

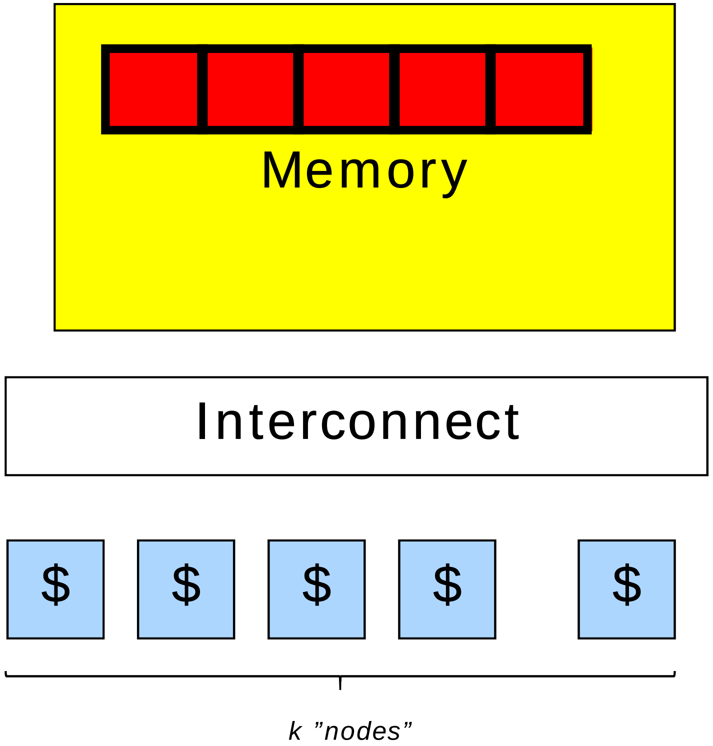
You can often increase the data set size with more processors

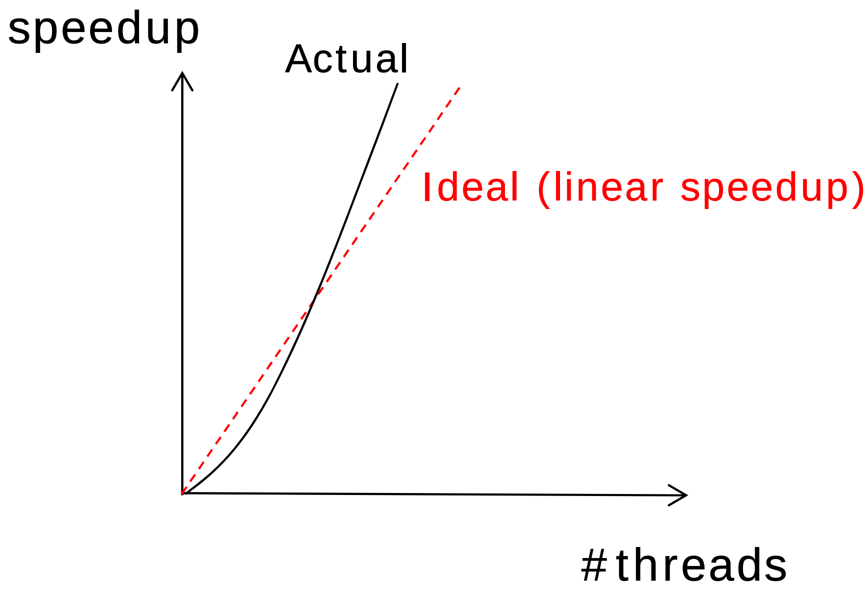
Strong scaling → Amdahl’s Law (same dataset)

Weak scaling → Gustavsson’s Law (scale the data set with P)

UPPSALA UNIVERSITET

Super-linear speedup





What kind of scaling will most likely see super-linear speedup?

- ❑ Strong scaling
- ❑ Weak scaling
- ❑ Mixed throughput workloads

Case Study: Gauss-Seidel on Multicores 1(2) Improving Locality

Erik Hagersten
Uppsala University

From Wallin et al, ICS 2006

(Longer explanation: **Finding a Door in the Memory Wall** @ **HPCWire.com**)

UPPSALA UNIVERSITET

Criteria for HPC Algorithms

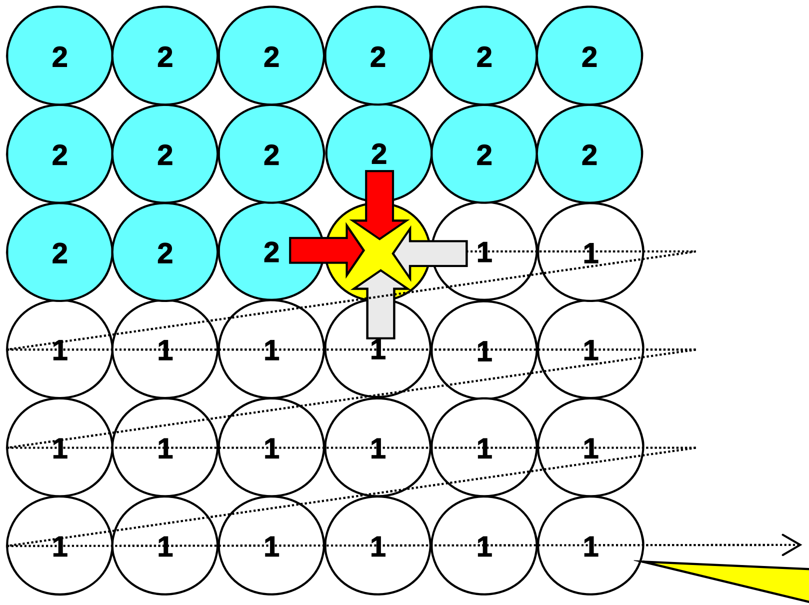
- Past:
  - Minimize communication
  - Maximize scalability (1000s of CPUs)
- Optimize for Multicore chip:
  - On-chip communication is “cheap”
  - Scalability need: ~10 – 20 threads
  - Caches capacity per thread is small
  - Memory latency/bandwidth a likely bottleneck/serialization component

➔ Data locality is important!

UPPSALA UNIVERSITET

Example: Gauss Seidel

<u>Mission:</u> "Maximize the parallelism and minimize the inter-thread communication"



- Reading old data
- Reading new data
- X Iterated X times
- Execution path
- Cell updated now

Often a huge data set (often three dimensions...). Will not fit in the cache

LOOP:
  UPDATE ALL POINTS WITH AVG FROM NEIGHBOURS
  IF (convergence_test)
    <done>

Data dependence → limited parallelism ☹

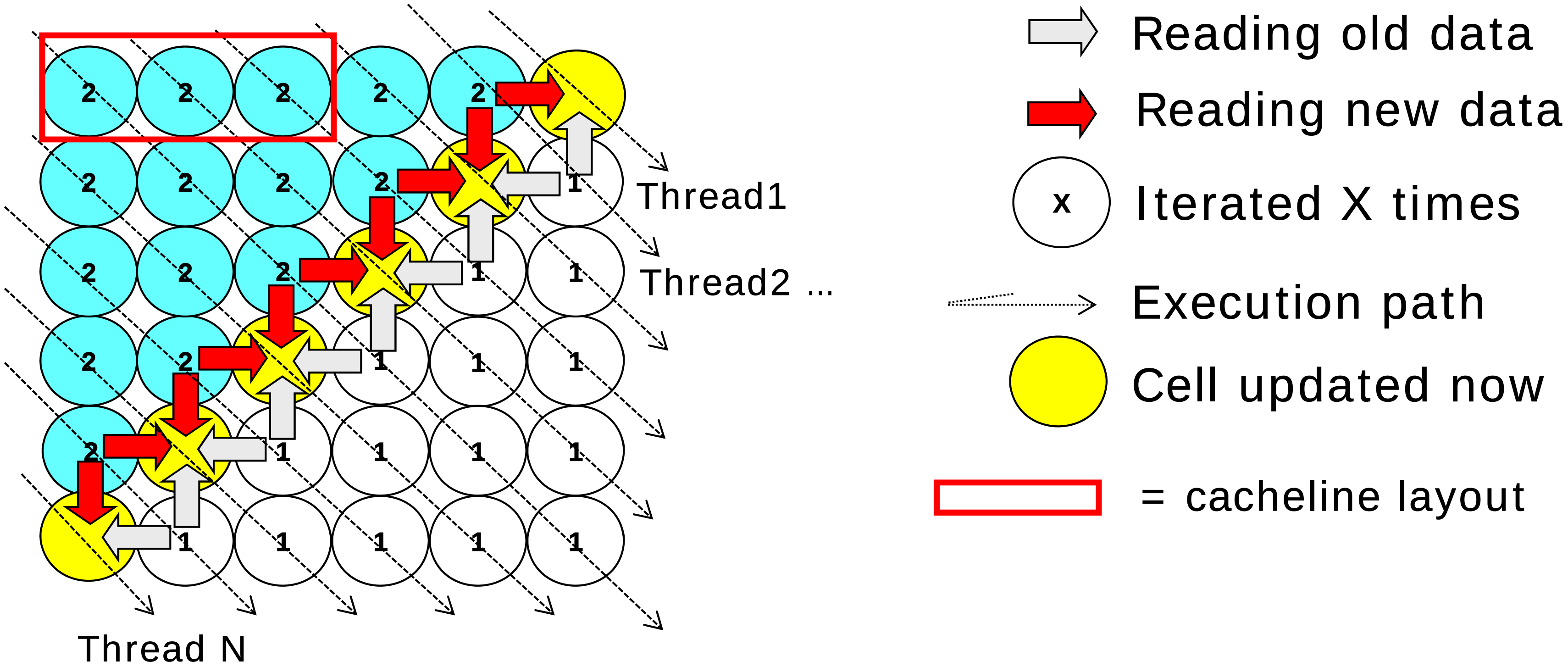
AVDARK 2013

UPPSALA UNIVERSITET

AVDARK 2013

Parallel Gauss Seidel

Threads synchronously handle one diagonal each
Wayfront needs to stay in sync (barriers)

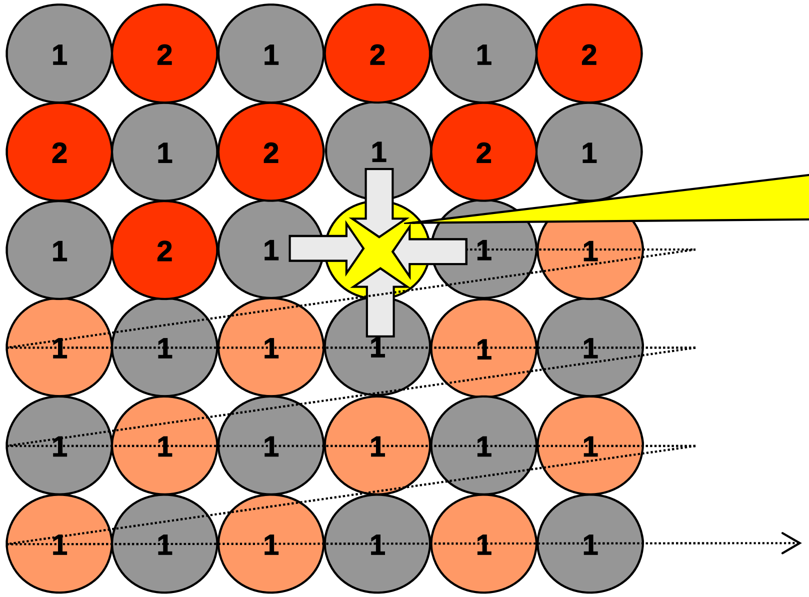


+ Parallelism N ☺
- Load imbalance ☹
- Synchronization needs ☹
- Data sharing ☹

State-of-the-art parallelization. Removing Dependencies: Red / Black

UPPSALA UNIVERSITET

Does not depend on any other concurrent updates



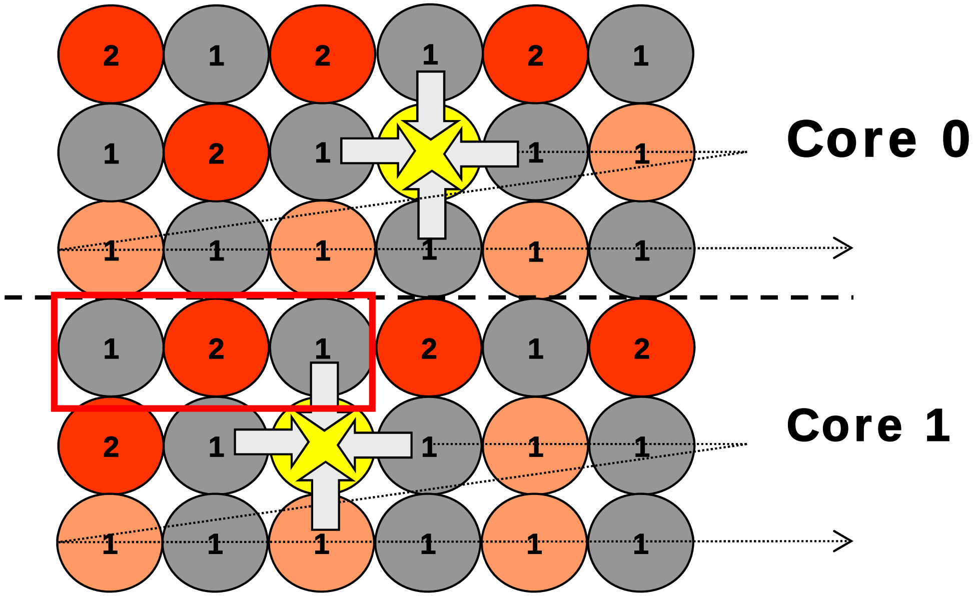
Different convergence due to different execution order

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

A
2013

UPPSALA UNIVERSITET

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

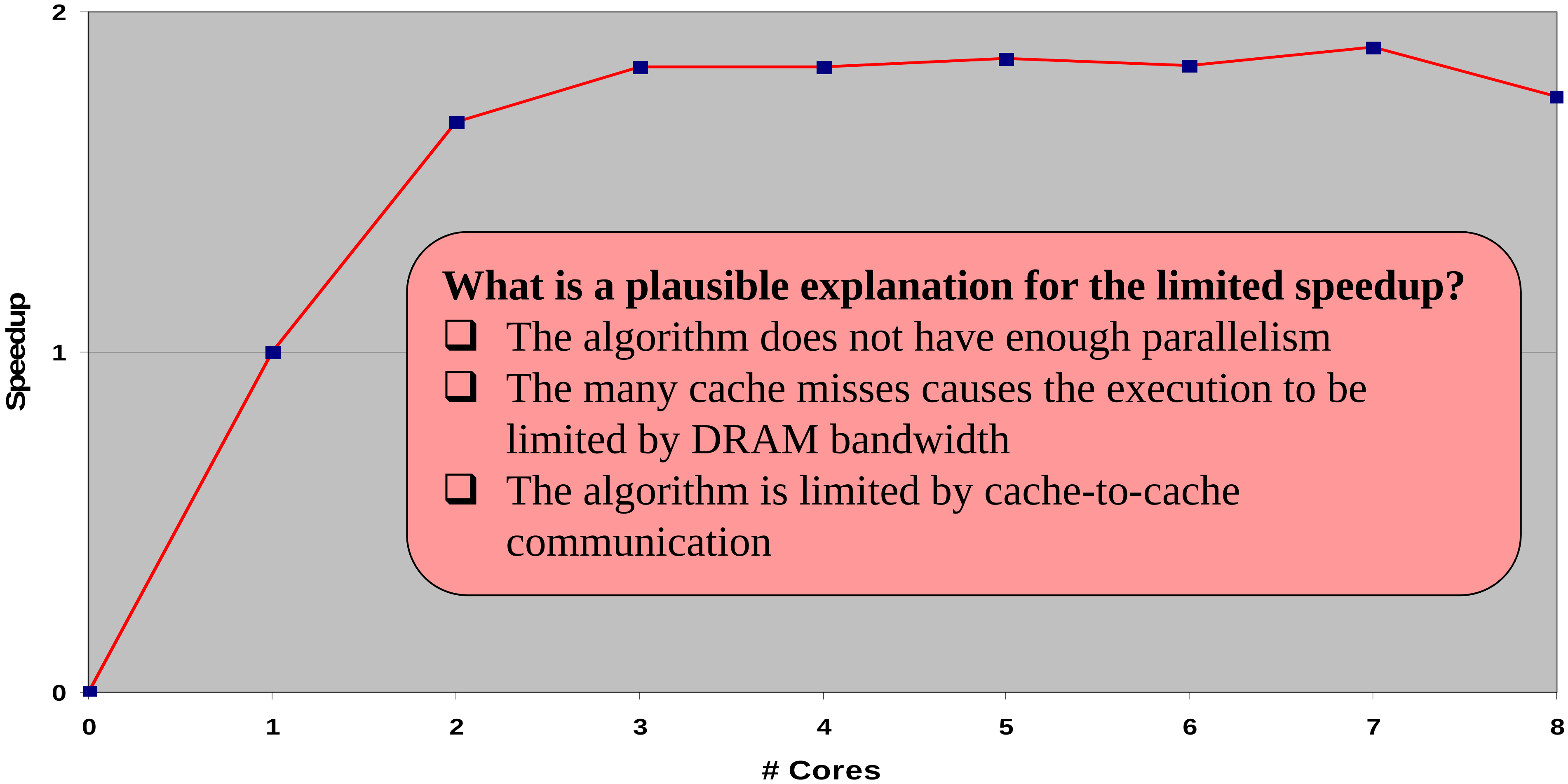


LOOP:
  IN PARALLEL: UPDATE ALL **RED** POINTS
  <barrier>
  IN PARALELL: UPDATE ALL **BLACK** POINTS
  <barrier>
  IF (convergence_test)
    <done>

$N^2/2$ parallelism ☺
Limited communication ☺
Done!
Only one problem…

UPPSALA UNIVERSITET

Only One Problem: Performance



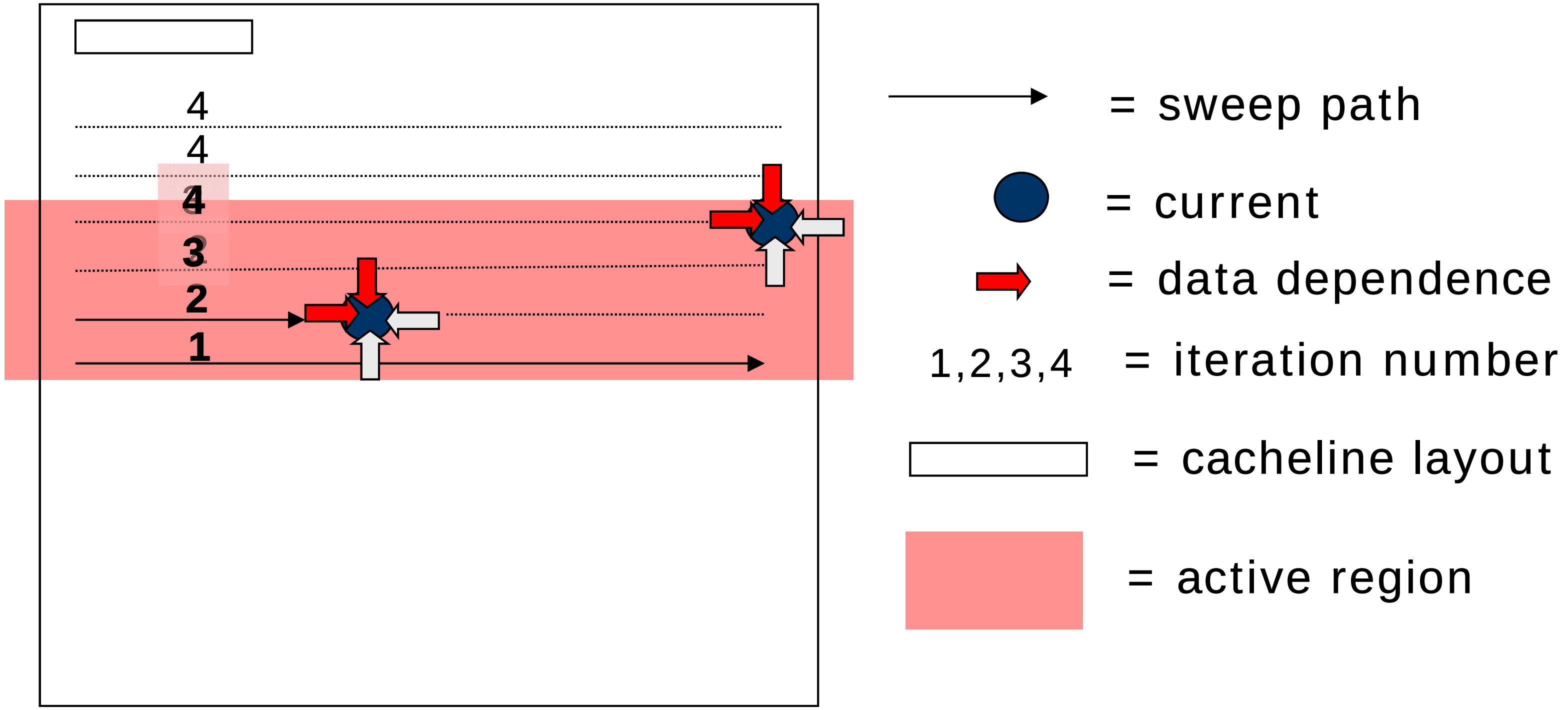
What is a plausible explanation for the limited speedup?

- ❑ The algorithm does not have enough parallelism
- ❑ The many cache misses causes the execution to be limited by DRAM bandwidth
- ❑ The algorithm is limited by cache-to-cache communication

Read / Black can double the number of cache misses
Sweeps the array twice per iteration

Back to the drawing board: Temporal blocking (here: seq. code)

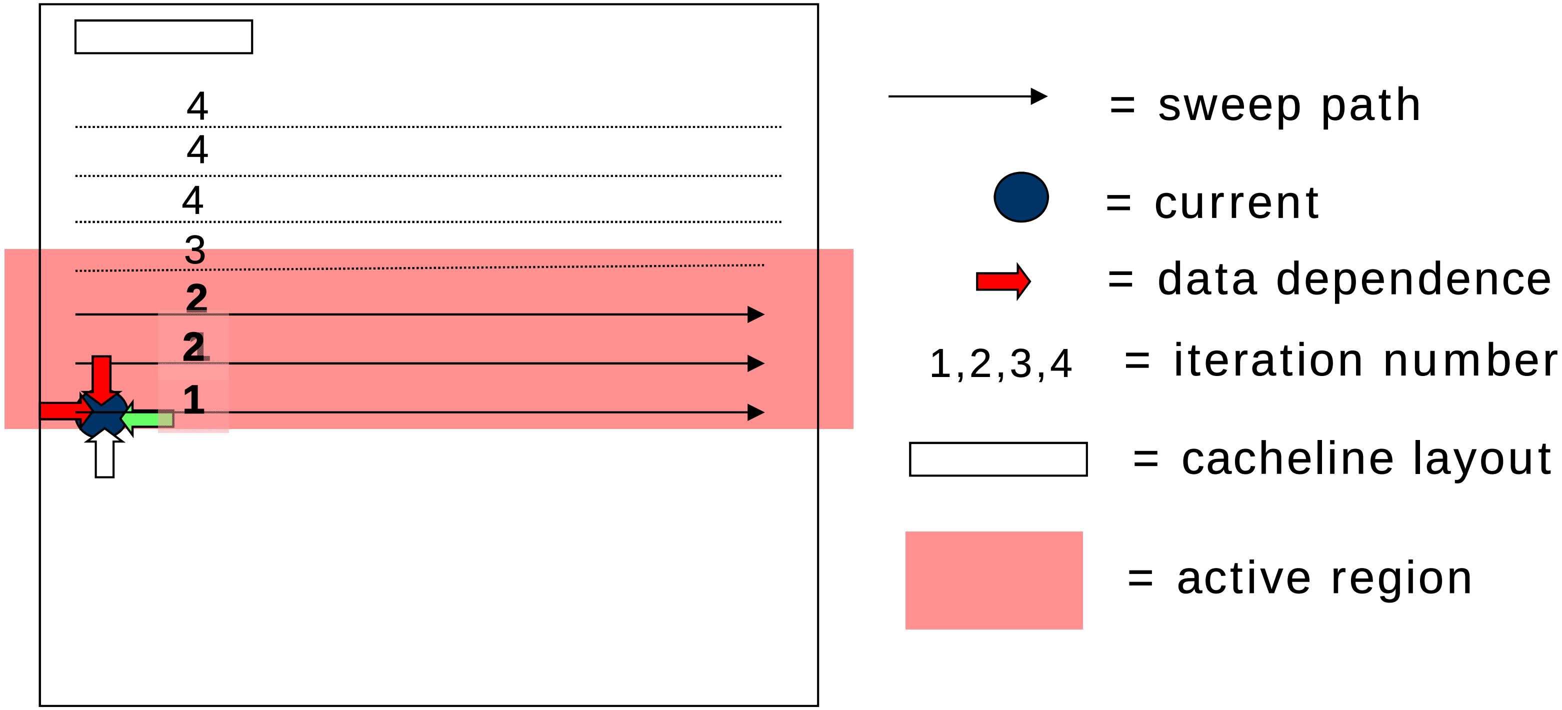
Communication is “cheap”. Moderate parallelism is OK
Priority 1: limit bandwidth needs!



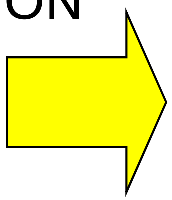
LOOP:
  LOOP OVER MAXTRIX
    UPDATE ALL POINTS IN ACTIVE REGION
    SLIDE DOWN THE REGION
  IF (convergence_test)
   <done>

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 OVER MATRIX
        UPDATE ALL POINTS IN ACTIVE REGION
        SLIDE DOWN THE REGION
    IF (convergence_test)
     <done>
```



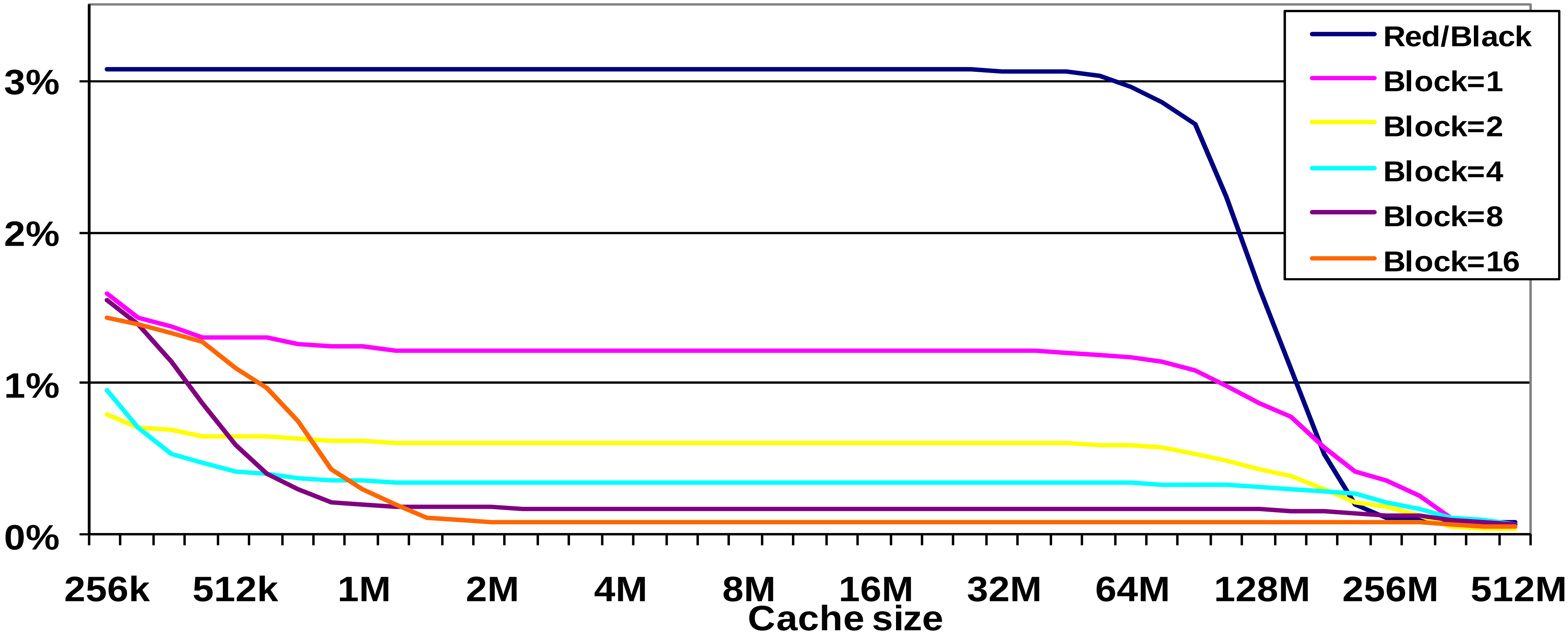
4 iterations in one sweep!

DRA

Larger blocks (active region) limits DRAM traffic for large caches, but seem to increase traffic for small caches. Why?

- ❑ The cache line size increase and cause more false sharing
- ❑ The size of the active region increases and cannot fit in the cache
- ❑ Spatial locality gets worse with larger blocks

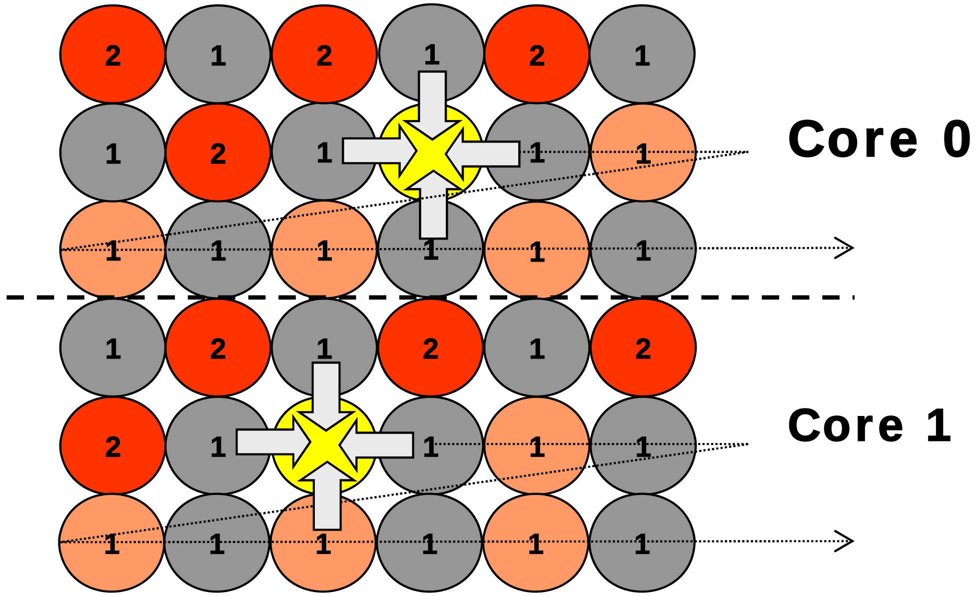
Fetch Ratio,
i.e, fraction of mem_ops generating DRAM traffic



Case Study:
Gauss-Seidel on Multicores
2(2) Adding Parallelism

From Wallin et al, ICS 2006

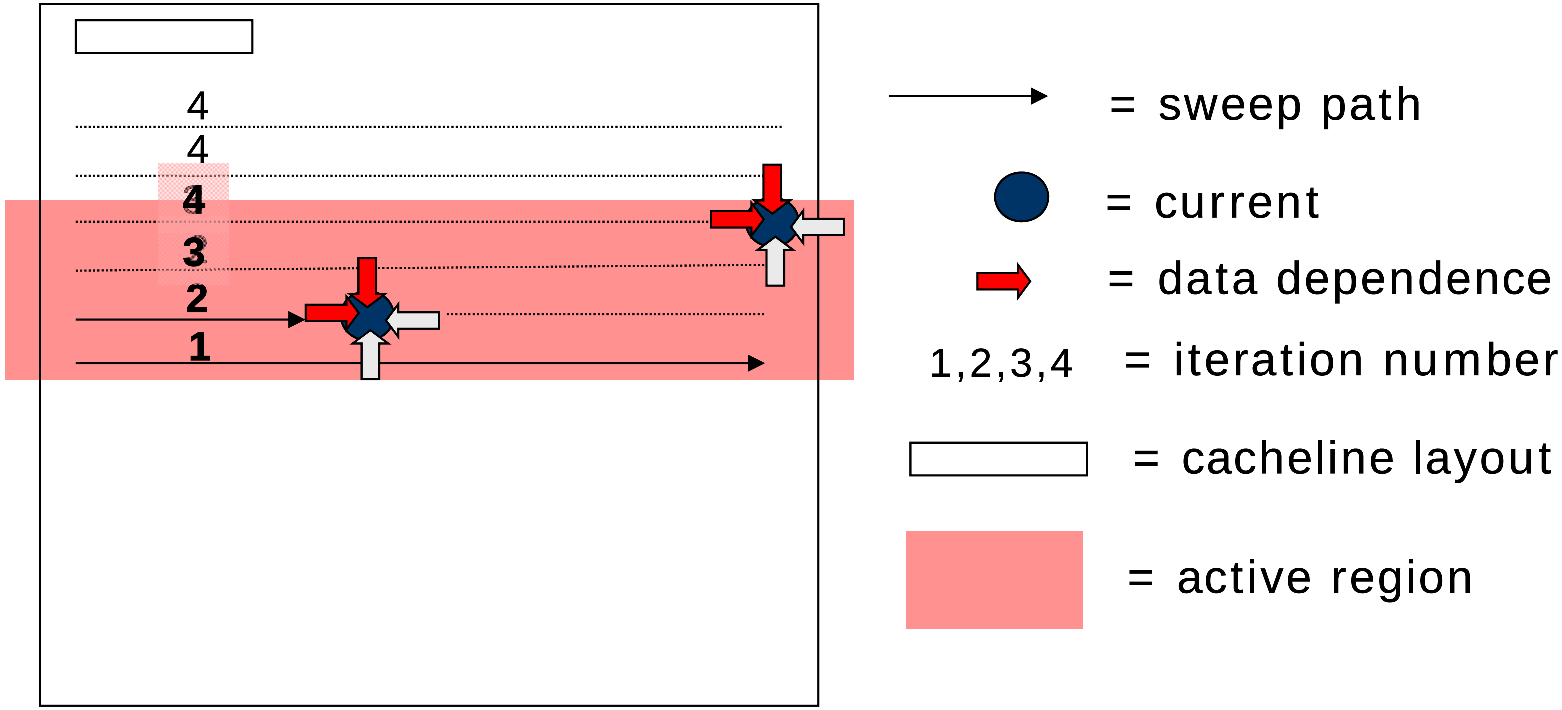
State-of-the-art parallelization Red/Black, Parallelism = $N^2/2$



LOOP:
  IN PARALLEL: UPDATE ALL **RED** POINTS
  <barrier>
  IN PARALELL: UPDATE ALL **BLACK** POINTS
  <barrier>
  IF (convergence_test)
    <done>

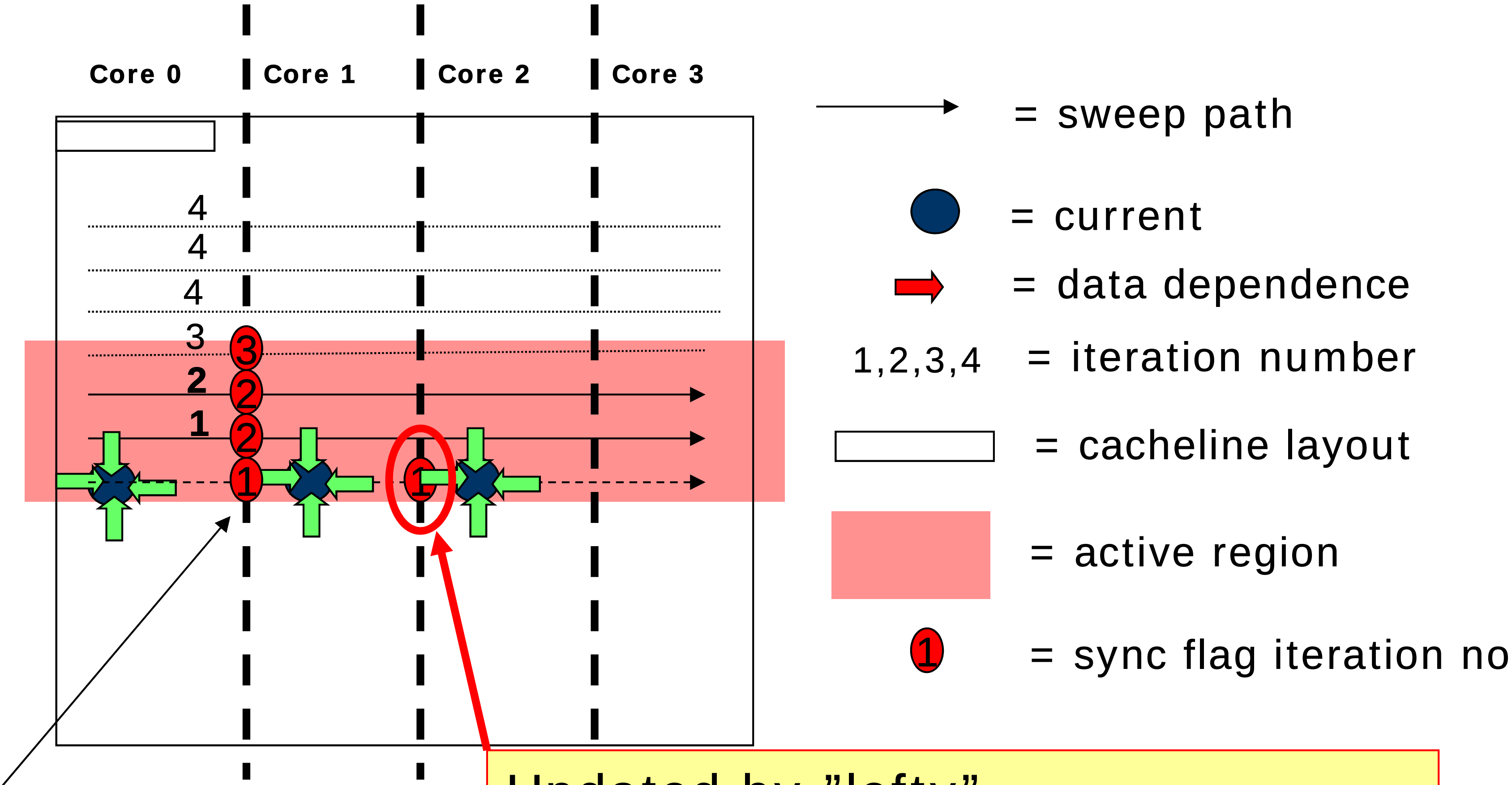
UPPSALA UNIVERSITET

We have data dependences (red arrows)




```
LOOP:
    LOOP:
        UPDATE ALL POINTS IN ACTIVE REGION
        SLIDE DOWN THE REGION
    IF (convergence_test)
     <done>
```

G-S, temp block Parallelism = N

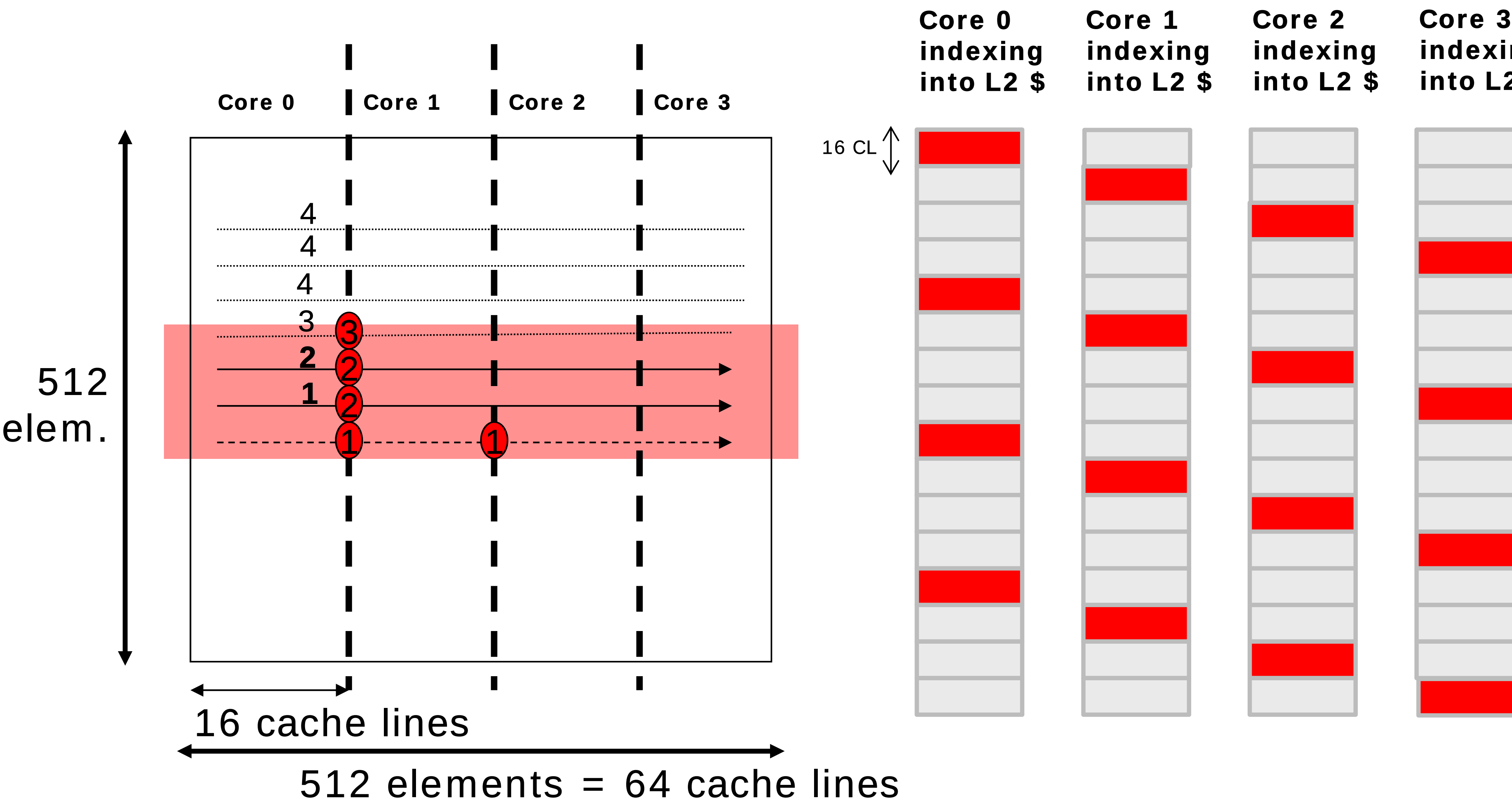


Synchronization flags

Updated by ”lefty”
“Righty”: 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)



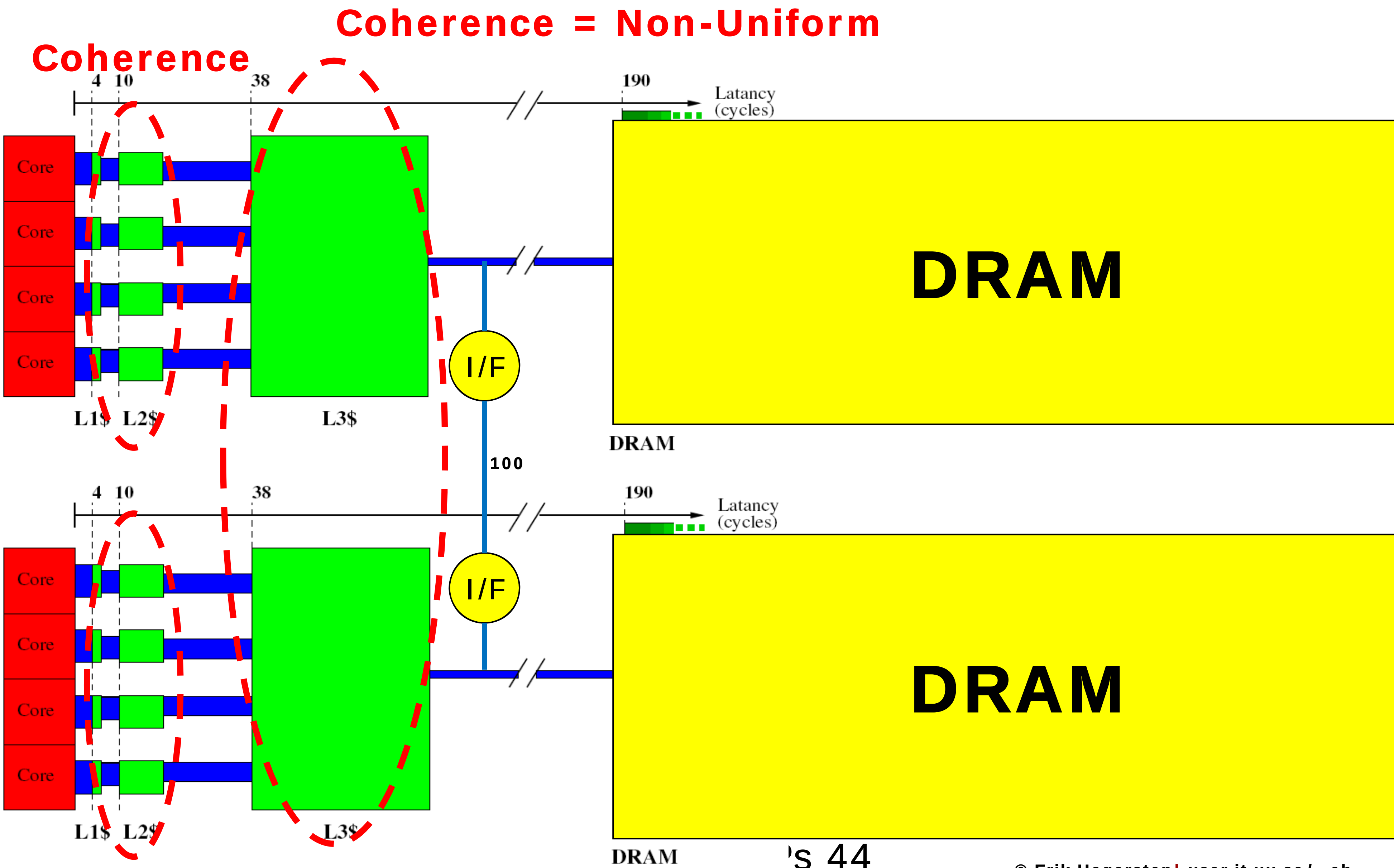
UPPSALA UNIVERSITET

Problems we ran into 2 (2)

- We had a loop nesting problem that the compiler optimized away
- … sometimes
- Flags were allocated too densly → False sharing

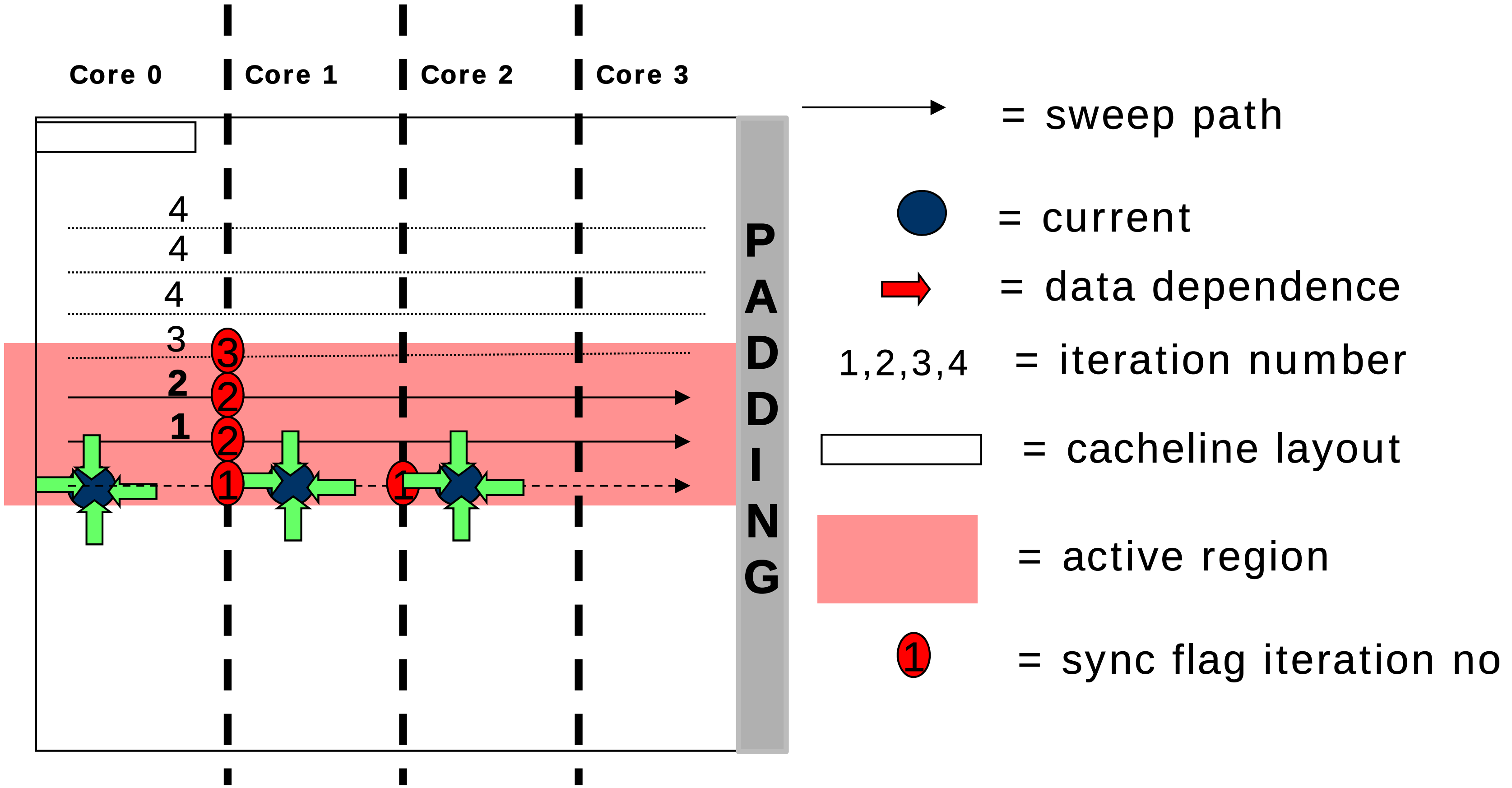
AVDARK 2013

Running on a Multisocket



UPPSALA UNIVERSITET

Example: G-S, temp blocking



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