

Tiling Loop Programs for Shared-Memory Multicore Architectures

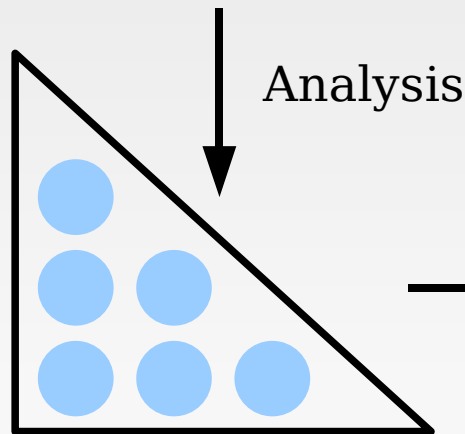
Armin Größlinger
Universität Passau
armin.groesslinger@uni-passau.de

Dagstuhl, 2007-09-04

The Polyhedron Model

- Mathematical model for loop programs
- Loops are described by polyhedra

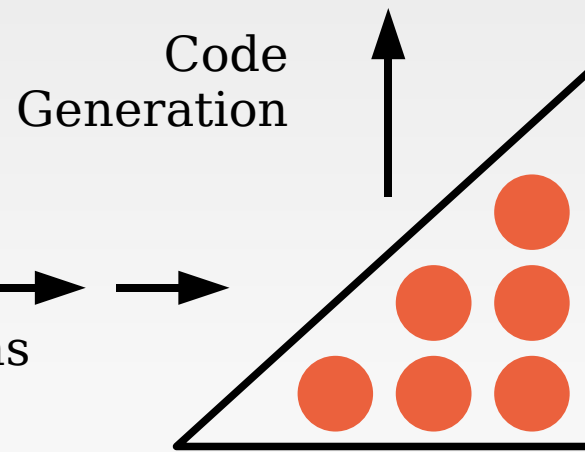
```
for i=0 to n
  for j=0 to n-i
    ...
```



$$0 \leq i \leq n$$
$$0 \leq j \leq n - i$$

Transformations

```
for i=0 to n
  for j=0 to i
    ...
```



$$0 \leq i \leq n$$
$$0 \leq j \leq i$$

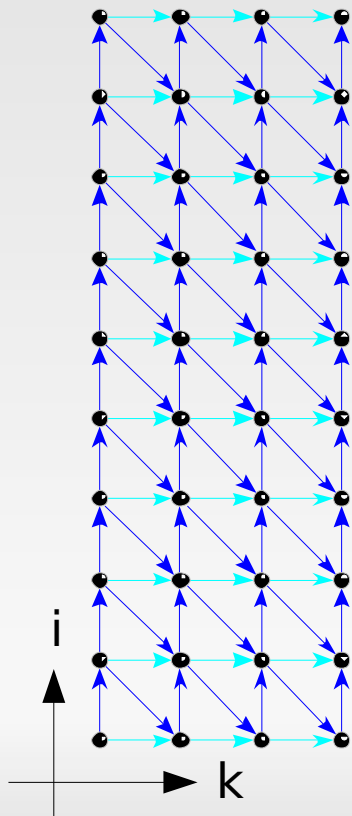
Our setting

- Input: sequential loop program
- Output: parallel program for SMP/NUMA architectures
- Main concerns:
 - Reduce fine-grained parallelism to available processors
 - Generate *efficient* target code
 - Take advantage of caches and memory locality

Example: 1D SOR

```
for k=1 to m
  for i=2 to n-1
    A[i] = (A[i-1] + A[i+1]) / 2
```

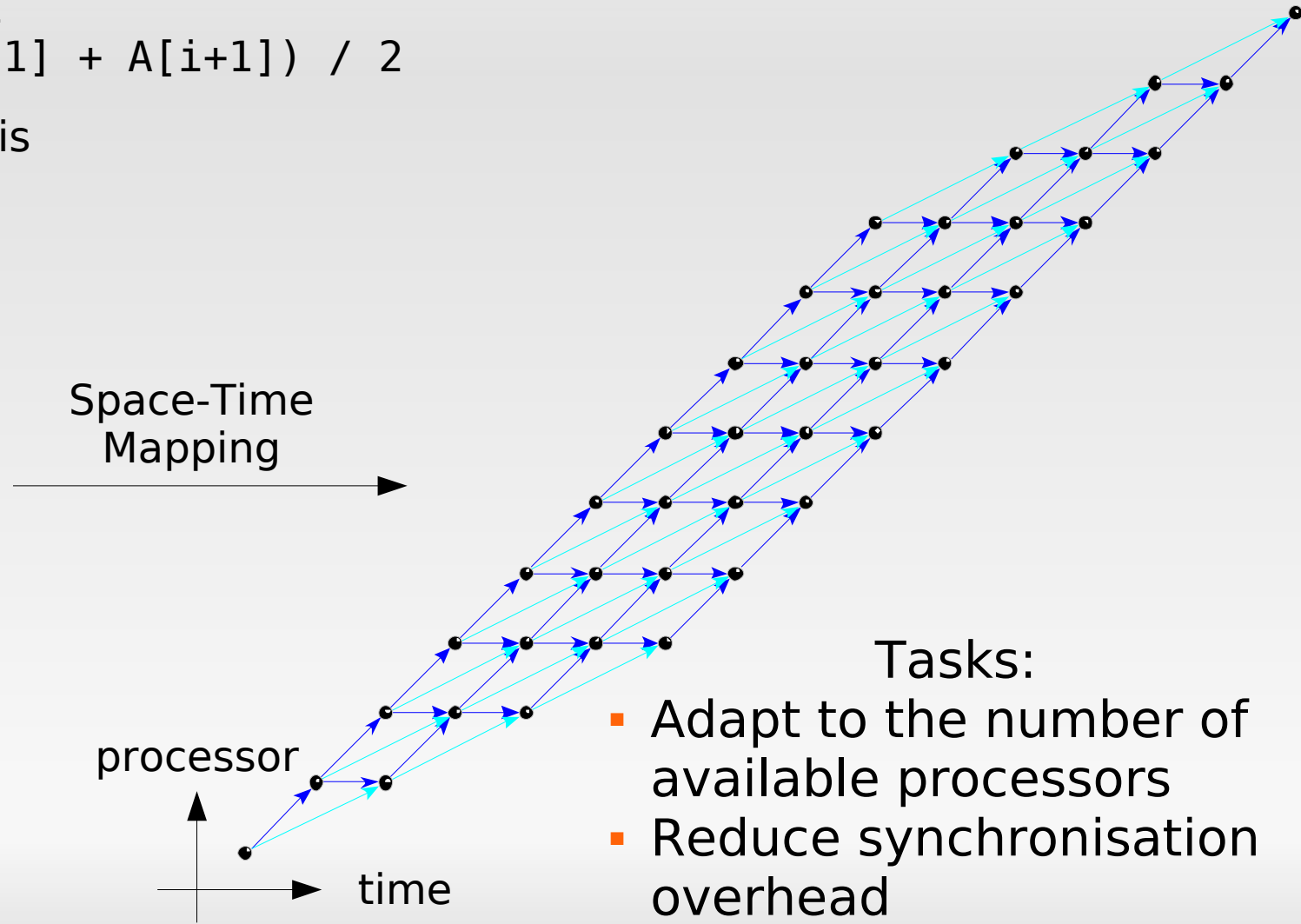
Analysis



Space-Time Mapping

processor

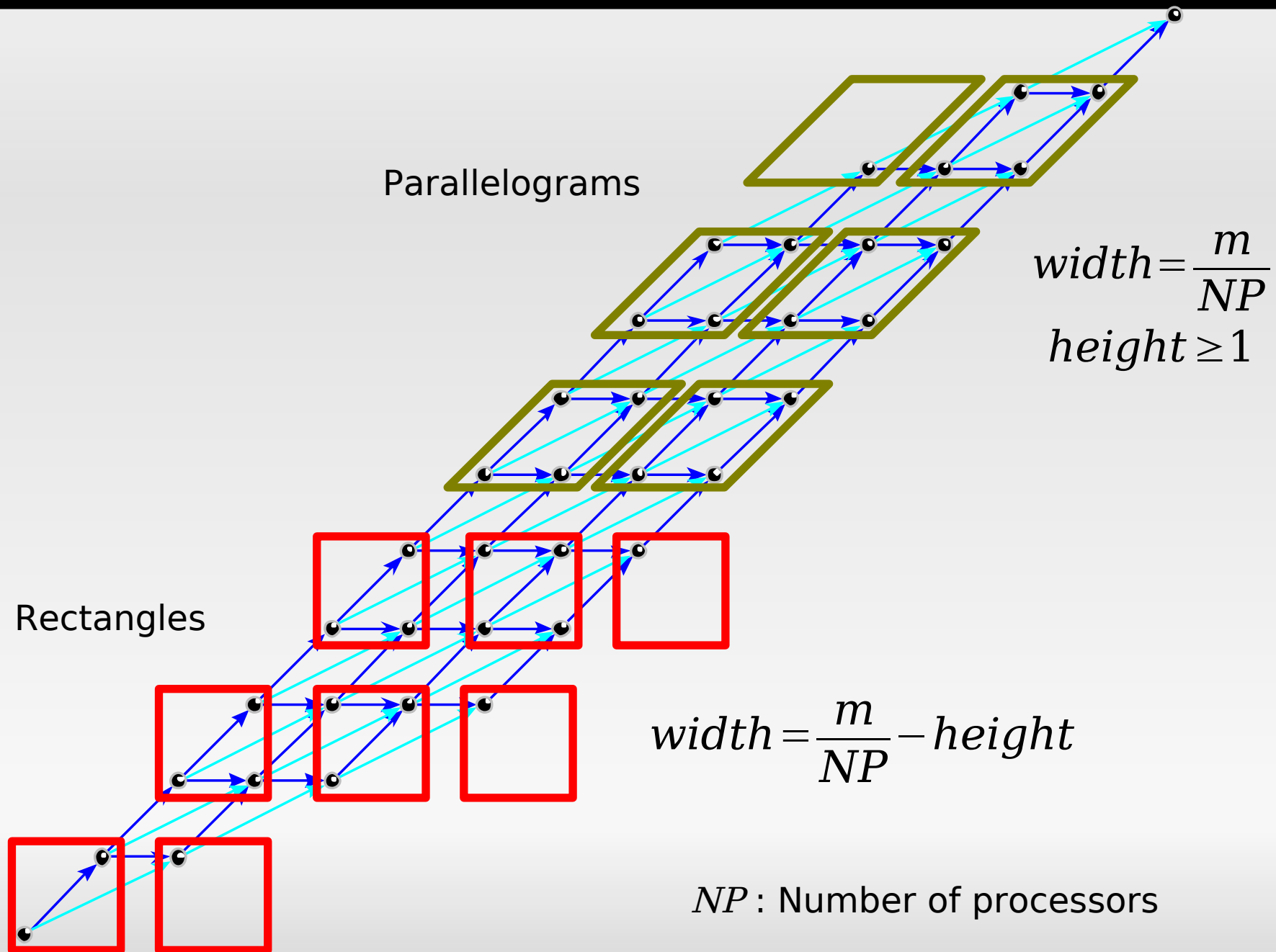
time



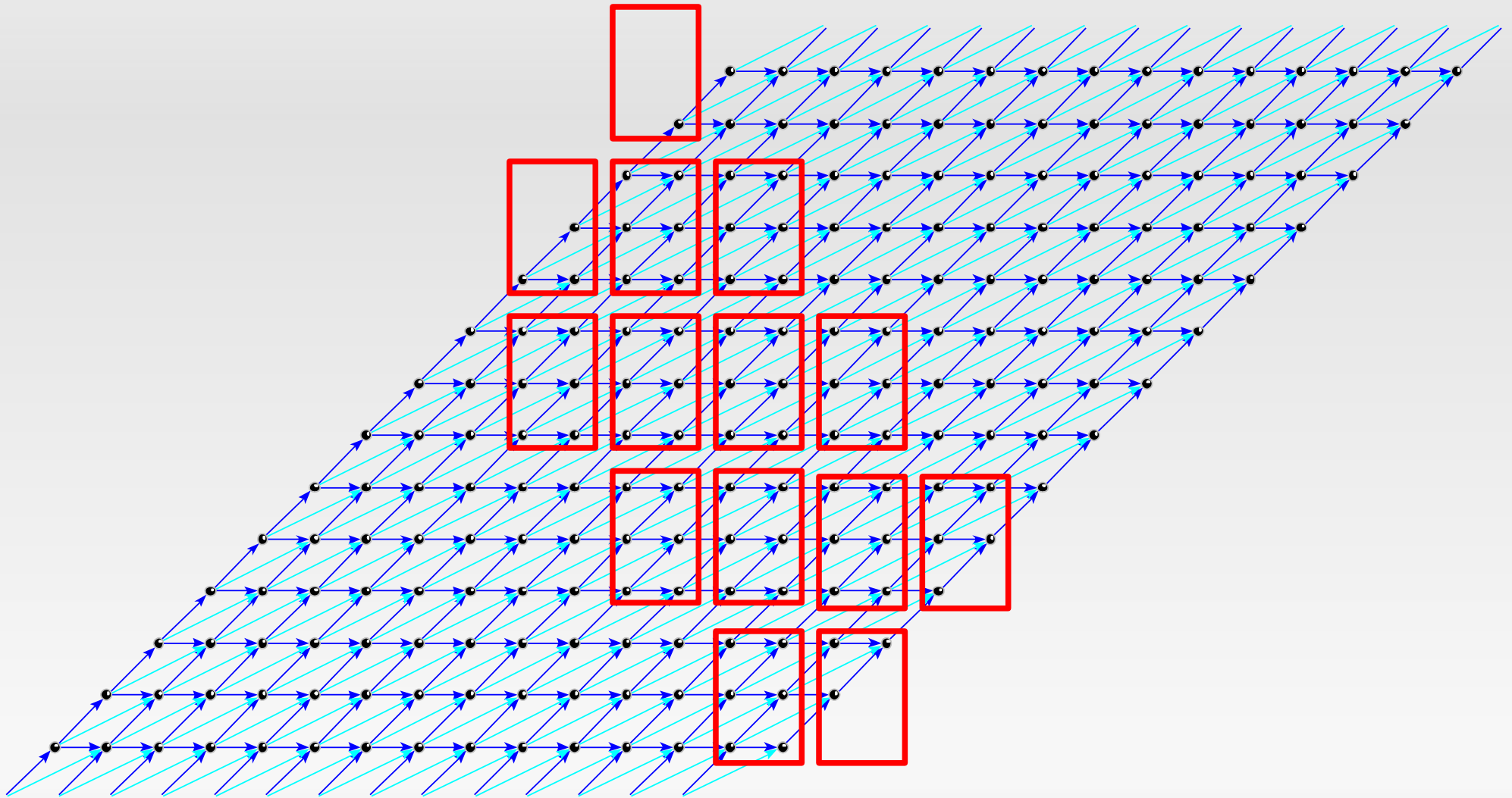
Tasks:

- Adapt to the number of available processors
- Reduce synchronisation overhead

1D SOR: 2 possible tilings



1D SOR: Rectangular tiling



1D SOR: Tiled code

Code is generated fully automatically using LooPo, CLoog, and M. Classen's code generation.
Manual part: selection of tile shape and size

```
#define S1(i, k) { A[i]=(A[1+i]+A[i-1])/2; }

int upperBound1 = floord(5*M+3*N-14,1500);
for (glT1=-1; glT1<= upperBound1; glT1++) {
    int upperBound2 = min(floord(1500*glT1+2*N+1493,7500),min(floord(M+N4,1500),
        min(floord(1500*glT1+1499,4500),floord(1500*glT1+1499,1500))));
    #pragma omp parallel for schedule(static,1) private(vT1,vP1)
    for (rp1=max(ceild(375*glT1-1499,1875),max(ceild(750*glT1-M-1498,2250),
        max(ceild(750*glT1-2*M-N+5,750),0))); rp1<= upperBound2; rp1++) {
        int upperBound3 = min(floord(1500*glT1+1500*rp1+1499,2),min(3000*rp1+2998,
            min(1500*rp1+M+1498,2*M+N-5)));
        for (vT1=max(750*glT1-750*rp1,max(3000*rp1-N+3,max(1500*rp1,0)));
            vT1<= upperBound3; vT1++) {
            int upperBound4 = min(1500*rp1+1499,min(floord(vT1+N-3,2),vT1));
            for (vP1=max(1500*rp1,max(ceild(vT1,2),vT1-M+1)); vP1<= upperBound4; vP1++) {

                S1(vT1-vP1+1, -vT1+2*vP1+2);

            }
        }
    }
}
}
```

1D SOR: Experiments

- Execution overhead of tiling for different tile sizes
- Cache behaviour
- Speed-up on 2x Dual-Core AMD Opteron

Parameters: $n=1,000,000$; $m=9,000$; $NP=4$

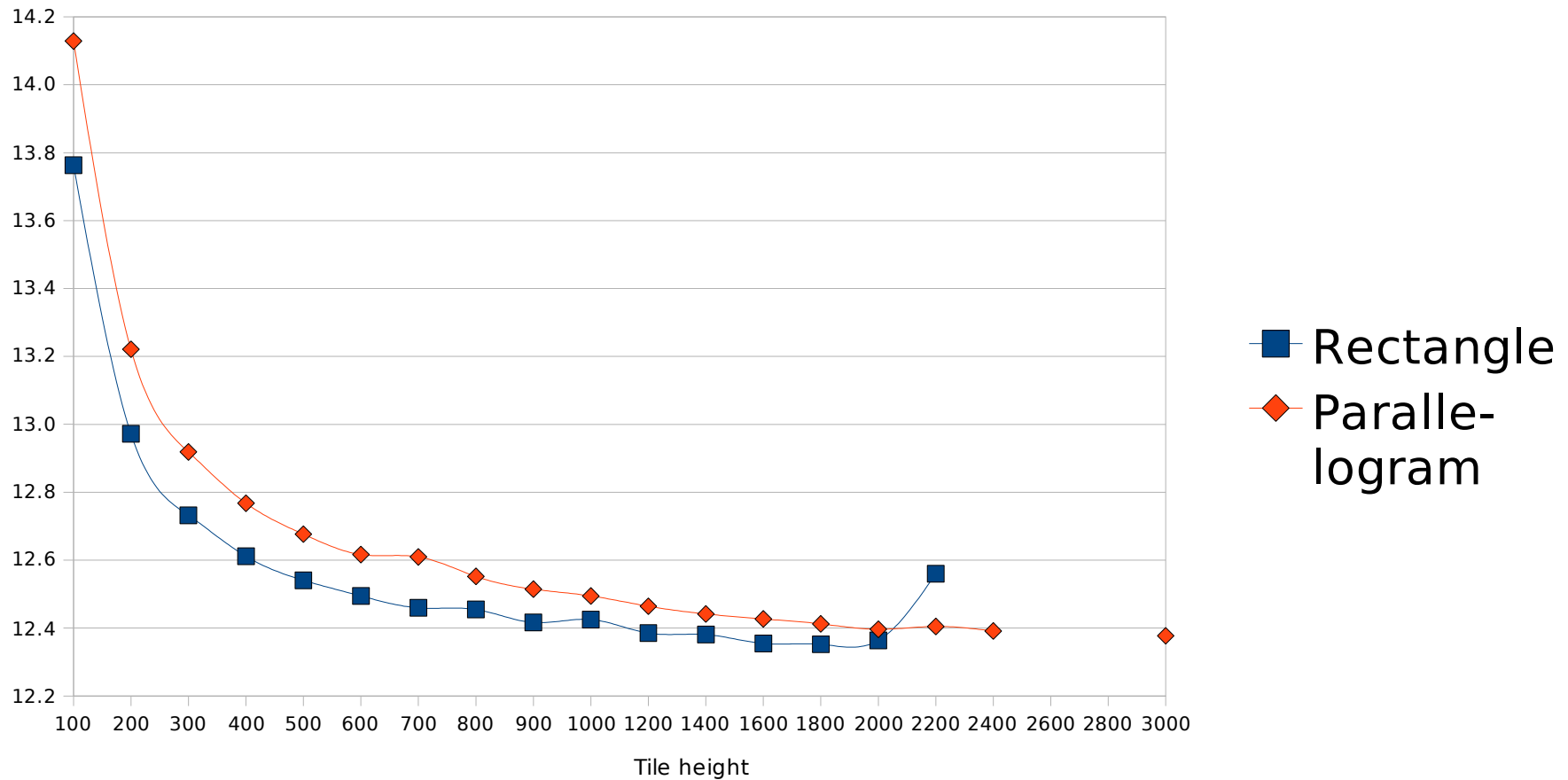
$$\Rightarrow m/NP = 2250$$

Rectangular: width = 2250 – height

Parallelogram: width = 2250

1D SOR: Tiling overhead

Execution time in seconds (on 1 processor)



1D SOR: Cache behaviour

Cachegrind results:

Instructions	Reads	Misses	Writes	Misses
18k	16k		2	
102k	88k			
42,235k	21,099k		18k	
36,056,066k	14,035k			
35,999,928k	17,999,964k	3,973k	8,999,982k	0

```
for glT1=  
  for rp1=  
    for vT1=  
      for vP1=  
        A[i]=(A[i-1]+A[i+1])/2
```

Code generated by GCC 4.2.1:

.L87:

```
fldl    (%eax)  
addl    $1, %edx  
faddl   -16(%eax)  
fmul    %st(1), %st  
fstpl   -8(%eax)  
addl    $16, %eax  
cmpl    %edx, %ecx  
jge     .L87
```

Complex loop bounds do not hurt performance if the innermost loop is not too short.

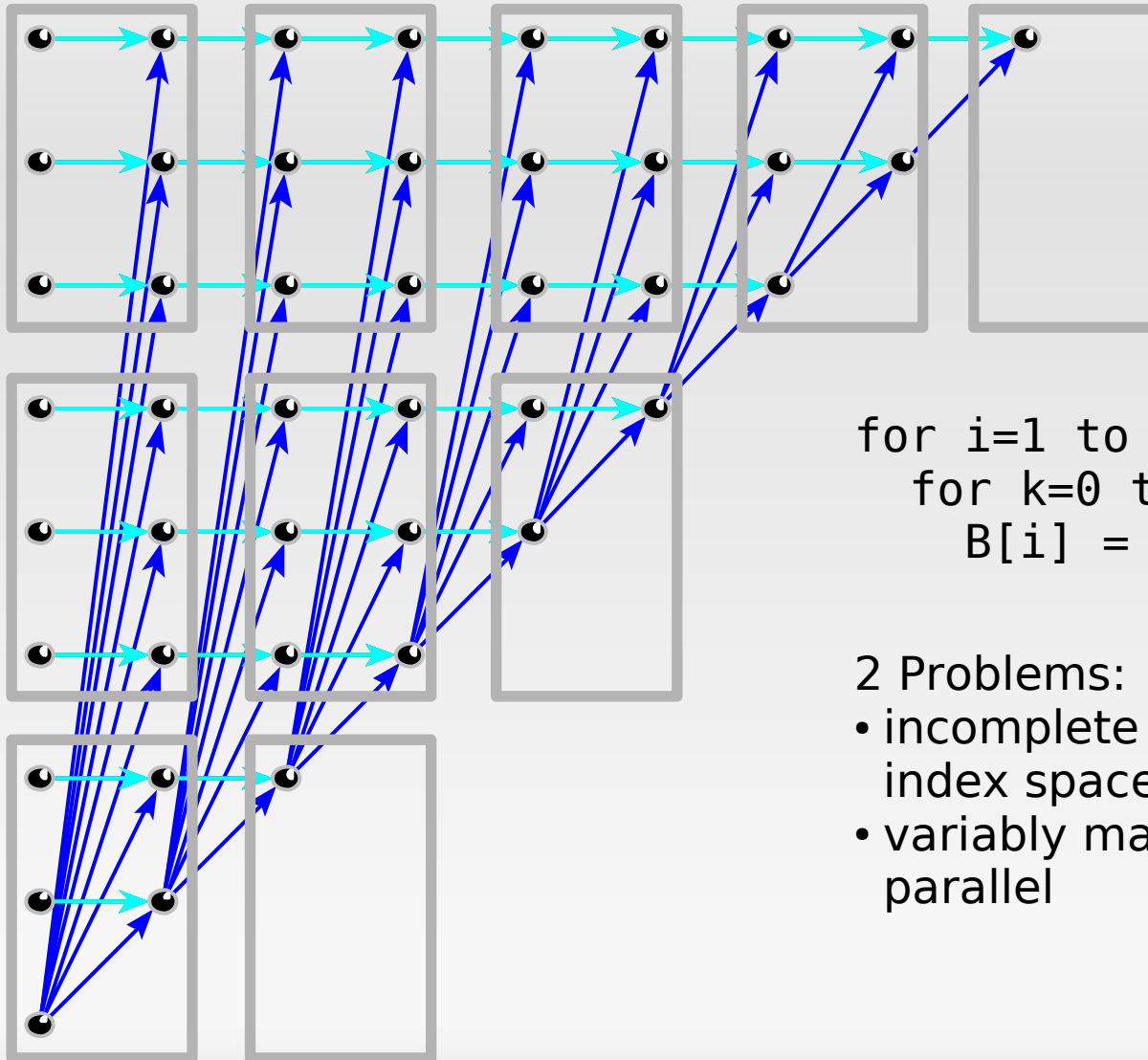
1D SOR: Speed-up

Processors	Sequential	Rectangular Tiling			Parallelogram Tiling		
		1	2	4	1	2	4
Time in secs	75.4 / 23.6	Width=850, Height=1400			Width=2250, Height=3000		
Speed-up		12.38	6.33	<u>3.23</u>	<u>12.38</u>	6.28	<u>3.17</u>
Efficiency*		1.00	1.95	3.82	1.00	1.97	3.90
		99.76%	97.55%	95.59%	99.76%	98.33%	97.40%
Time in secs		Width=650, Height=1600			Width=2250, Height=3000		
Speed-up		<u>12.35</u>	6.32	3.26	<u>12.38</u>	6.28	<u>3.17</u>
Efficiency*		1.00	1.95	3.79	1.00	1.97	3.90
		100.00%	97.71%	94.71%	99.76%	98.33%	97.40%

systematic?

2x Dual-Core AMD Opteron 2.2 Ghz

Example: Backward subst.

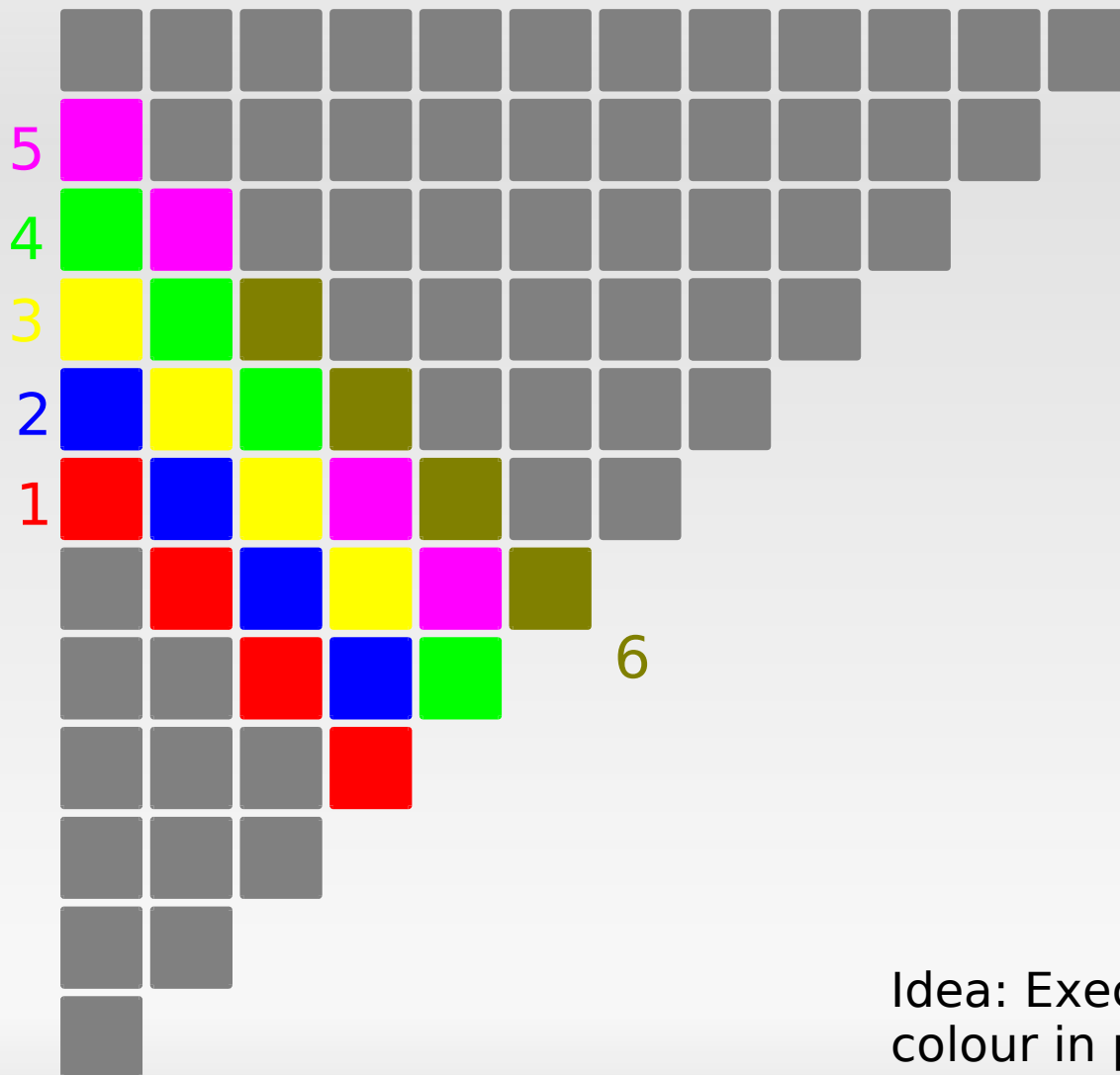


```
for i=1 to n-1
  for k=0 to i-1
    B[i] = B[i] - A[i][k]*B[k]
```

2 Problems:

- incomplete tiles on *one* side of the index space
- variably many tiles to execute in parallel

BwSub: Cyclic tile distribution



Idea: Execute tiles of the same colour in parallel on 4 processors

BwSub: Experiment

	Processors		Real Speed-up	Theoretical Speed-up in a very simple model
	1	2		
Sequential	203ms	-		
Simple parallel execution	177ms	124ms	1.43	1.832
Cyclic parallel execution	177ms	140ms	1.26	1.829

Why this difference?
Cache behaviour is very similar.

Dual-Core AMD Opteron 2.2 Ghz
n=10,000

Conclusion/Outlook

- 1D SOR and other examples suggest that the polyhedron model can deliver scalable parallel programs
- Parameters (tile size etc.) must be carefully chosen (how?)
- Code generation still a hard problem
- Future directions:
 - Increase applicability of the model
 - Computation and/or data replication?