

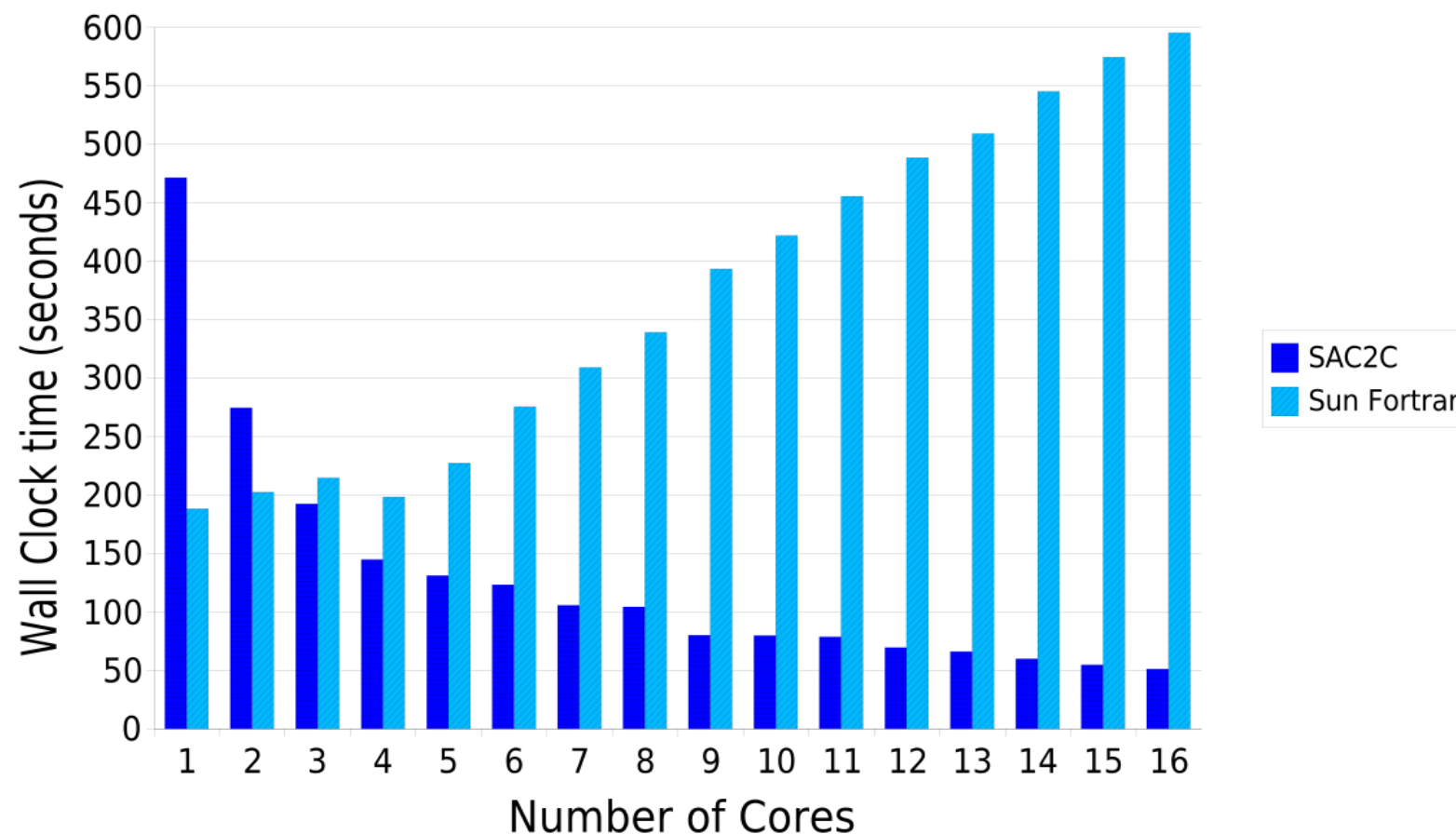
Advanced Programming

HPC Code Generation

Peter Achten, Sven-Bodo Scholz

Software Science
Radboud University Nijmegen

SMP Performance: KVD Shockwave Simulations

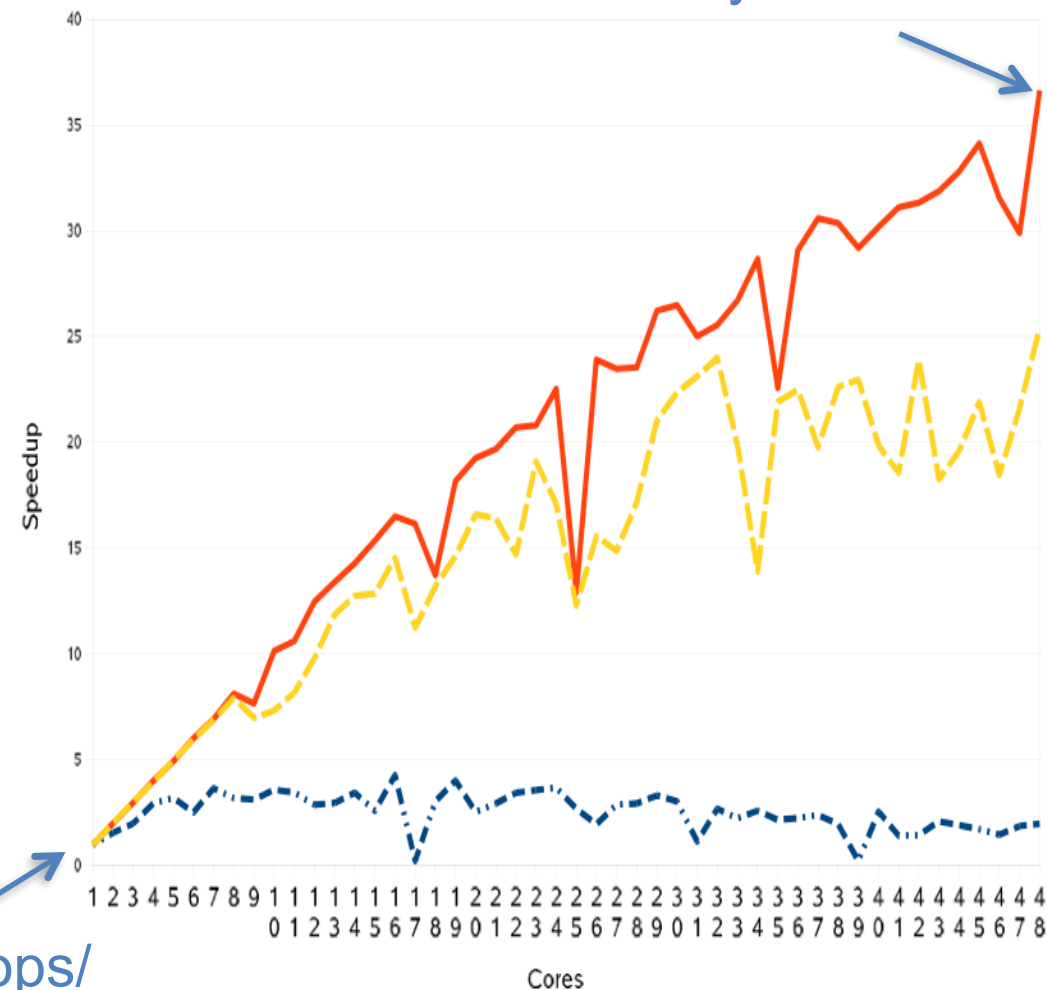


[Sch03] Sven-Bodo Scholz. Single Assignment C - efficient support for high-level array operations in a functional setting.
Journal of Functional Programming, 13(6):1005-1059, 2003.

Performance Portability: Image Smoothing

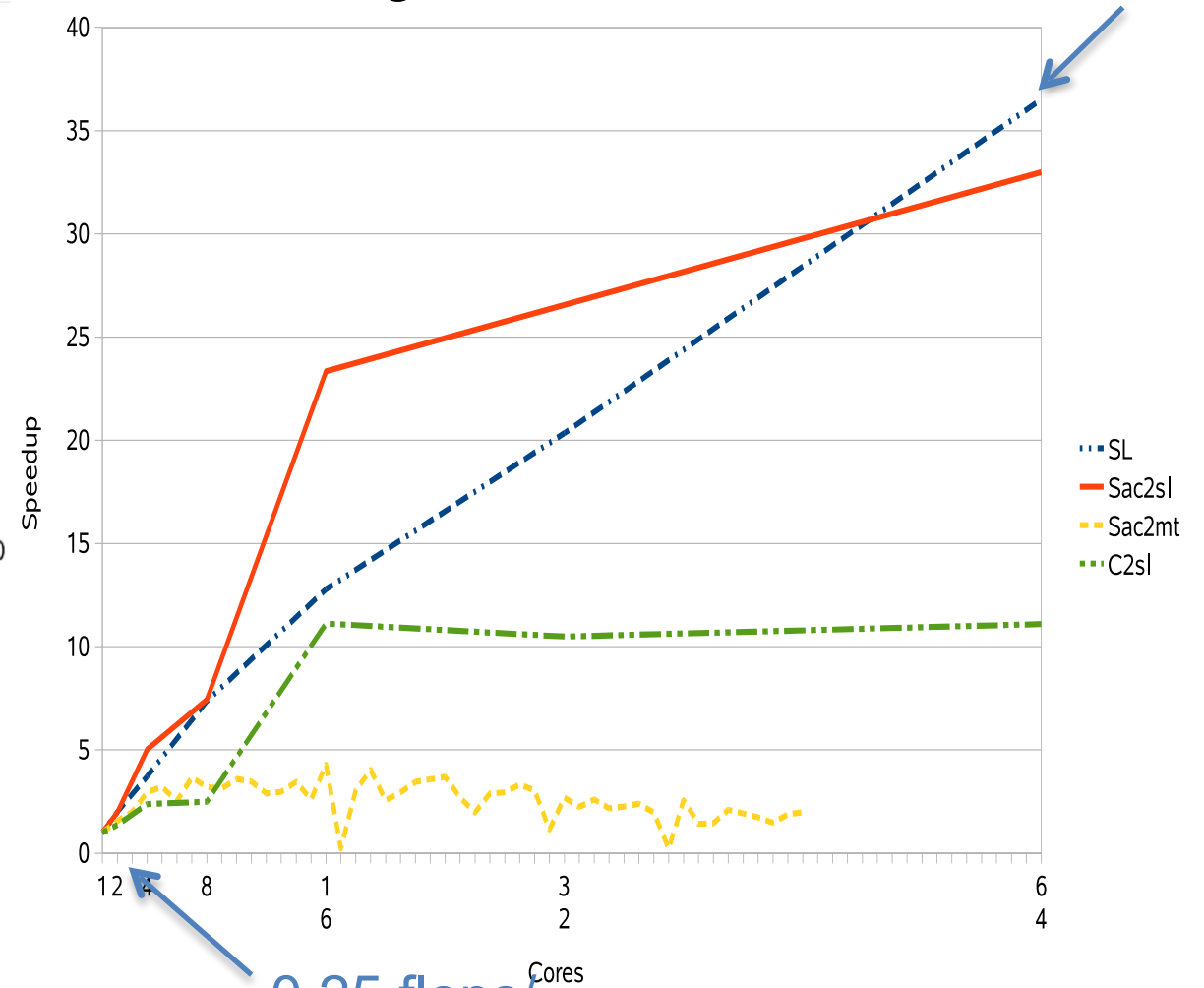
48 core SMT

23 flops/
cycle



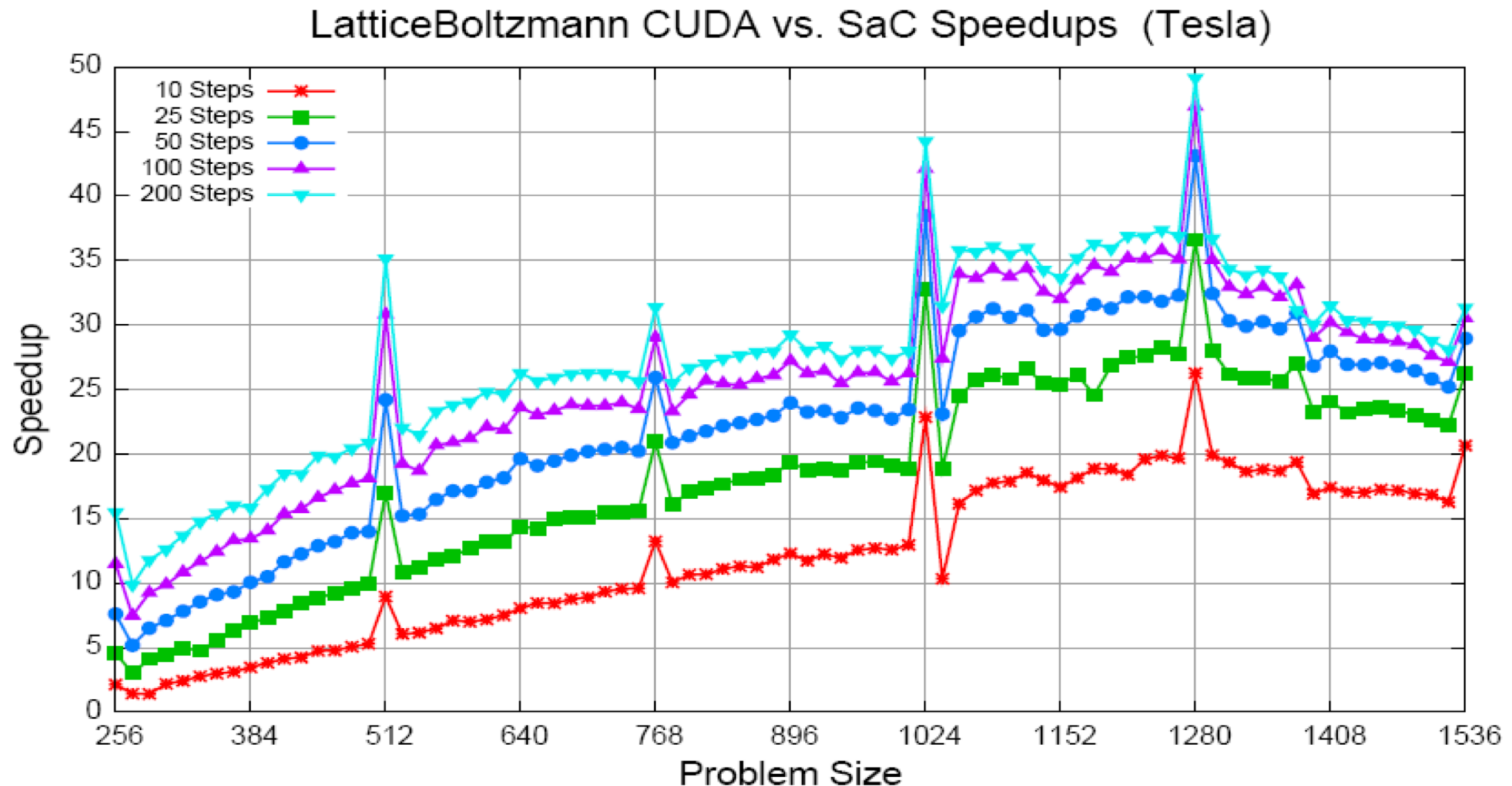
64 core Microgrid

8 flops/cycle



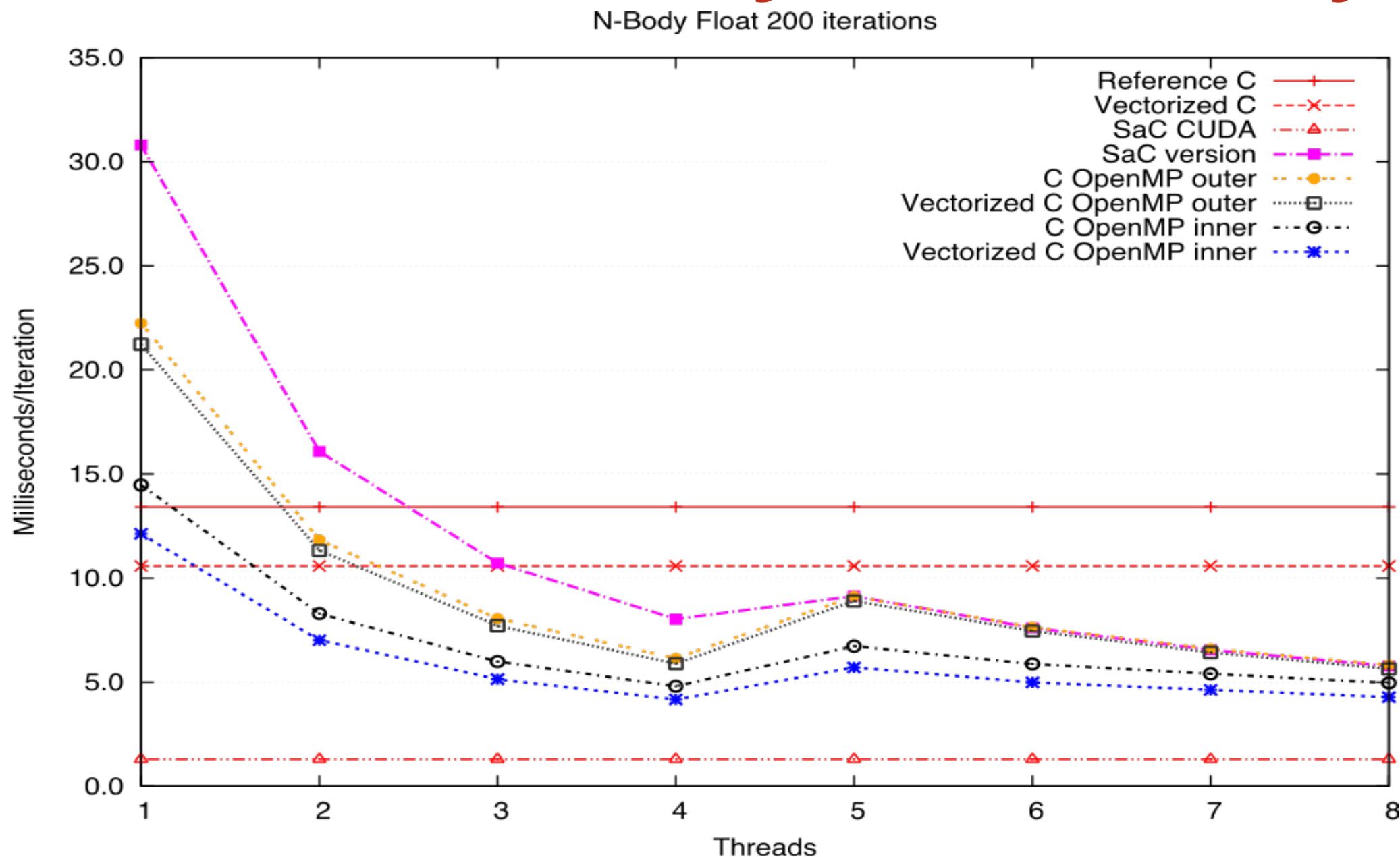
[HJS+11] S. Herhut, C. Joslin, S.B. Scholz, R. Poss, and C. Grelck. Concurrent Non-Deferred Reference Counting on the Microgrid: First Experiences. In J. Haage and M. Morazán, editors, *IFL'10, Revised Selected Papers*, LNCS 6647, pages 185-202. Springer, 2011.

GPU Performance: Lattice Boltzmann



[GTS11] Jing Guo, Jeyarajan Thiyagalingam, and Sven-Bodo Scholz. Breaking the GPU programming barrier with the auto-parallelising SAC compiler. In *DAMP'11, Austin, USA*, pages 15-24. ACM Press, 2011.

Performance Portability: Naive N-Body



[vSB⁺13] A. Šinkarovs, S.B. Scholz, R. Bernecky, R. Douma, and C. Grelck. SAC/C formulations of the all-pairs N-body problem and their performance on SMPs and GPGPUs. *Concurrency and Computation: Practice and Experience*, 2013

Anisotropic Diffusion

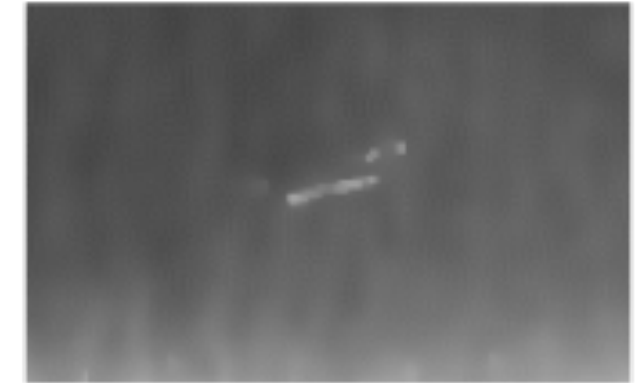
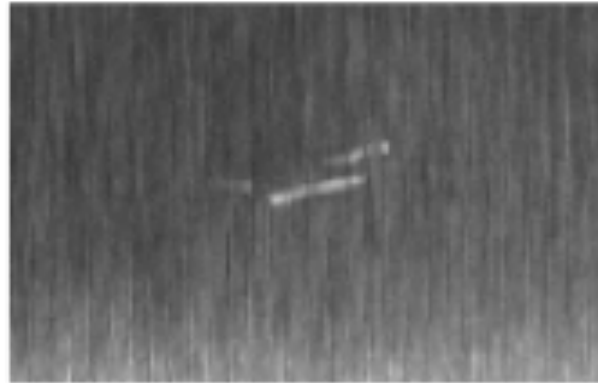


	image dimension				
	$px256 \times px256$	$px512 \times px512$	$px1024 \times px1024$	$px2048 \times px2048$	$px4096 \times px4096$
SAC-MT 1T	0.199 sec	0.788 sec	3.165 sec	12.55 sec	50.18 sec
SAC-MT 8T	0.067 sec	0.156 sec	0.635 sec	2.562 sec	10.18 sec
SAC-CUDA	0.003 sec	0.006 sec	0.015 sec	0.036 sec	0.181 sec
CUDA-manual	0.005 sec	0.007 sec	0.016 sec	0.050 sec	0.190 sec
OpenCV(1)	0.119 sec	0.540 sec	2.374 sec	9.571 sec	54.45 sec
OpenCV(2)	0.097 sec	0.550 sec	2.386 sec	9.507 sec	53.21 sec
OpenCV(3)	0.121 sec	0.512 sec	2.298 sec	9.532 sec	53.89 sec
OpenCV(4)	0.098 sec	0.498 sec	2.321 sec	9.532 sec	54.10 sec
Matlab	0.164 sec	0.608 sec	2.518 sec	10.06 sec	438.6 sec

[WGH⁺12] V. Wieser, C. Grelck, P. Haslinger, J. Guo, F. Korzeniowski, R. Bernecky, B. Moser and S.B. Scholz. Combining high productivity and high performance in image processing using Single Assignment C on multi-core CPUs and many-core GPUs. *Journal of Electronic Imaging*, 21(2), 2012.



The Beauty of Generality

```
int[n:shp] add (int[n:shp] a, int[n:shp] b)
{
    return { iv -> a[iv] + b[iv] };
}
```

specialise

specialise

```
int[...]  
add( int[...] a, int[...] b)
{
    res = with {
        ( 0*shape(a) <= [i,k] < shape( a)) : a[[i,k]] + b[[i,k]];
    } : genarray( shape( a), 0);
    return res;
}
```

```
int[3,5] add( int[3,5] a, int[3,5] b)
{
    res = with {
        ( [0,0] <= [i,k] < [3,5]) : a[[i,k]] + b[[i,k]];
    } : genarray( [3,5], 0);
    return res;
}
```


The Beast of Generality

```
int[n:shp, m:ishp] take (int[n] shp, int[n:oshp, m:ishp] a)
{
  return { iv -> a[iv] | iv < shp };
}
```

specialise

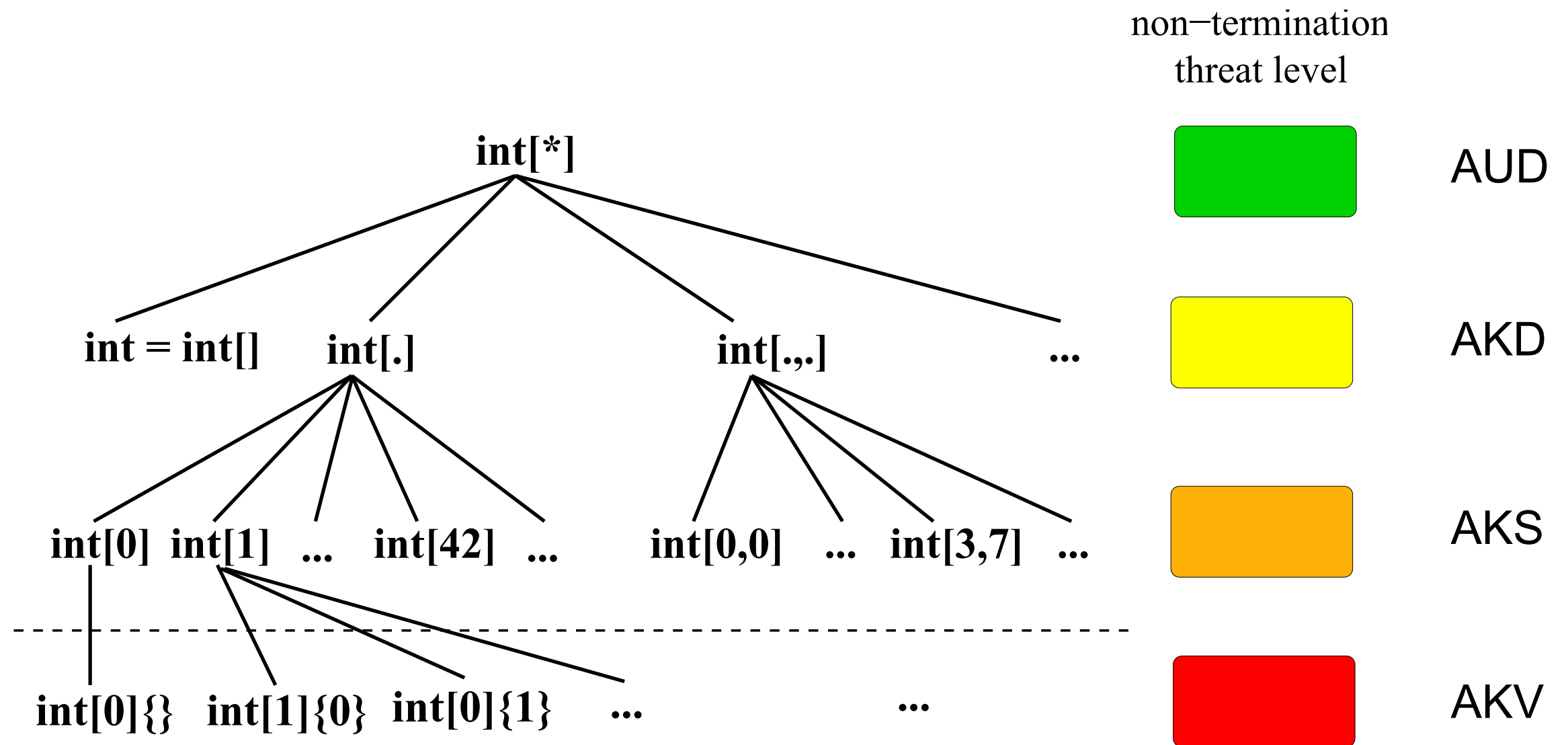
specialise



```
int[.,.] take( int[.] shp, int[.,.] a)
{
  res = with {
    ( 0*shp <= [i,k] < shp) : a[[i,k]];
  } : genarray( shp, 0);
  return res;
}
```

```
int[.,.] take( int[2] shp, int[3,5] a)
{
  res = with {
    ( 0*shp <= [i,k] < shp) : a[[i,k]];
  } : genarray( shp, 0);
  return res;
}
```

Pandora's box:



The Need for PE (partial evaluation) – part I

Insight I:

We want to control specialisation to avoid non-termination!

Insight II:

A specialisation oracle always exists
=> we can always minimise specialisation!

[GSS06] C.Grelck, S.B.Scholz, and A.Shafarenko. A Binding Scope Analysis for Generic Programs on Arrays. In Andrew Butterfield, editor, *IFL'05*, LNCS 4015, pp 212–230. Springer, 2006.

Take Revisited

```
int[n:shp, m:ishp] take (int[n] shp, int[n:oshp, m:ishp] a)
{
  retrun { iv -> a[iv] | iv < shp };
}
```

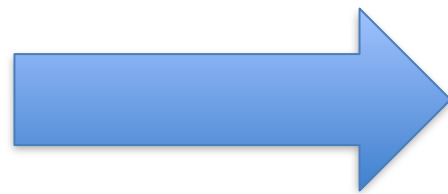
 specialise

```
Int[2,3] take( int[2]{2,3} shp, int[3,5] a)
{
  res = with {
    ( [0,0] <= [i,k] < [2,3] ) : a[[i,k]];
  } : genarray( [2,3], 0);
  return res;
}
```

res	shp	a
AUD	AUD	AUD
AKD	AKD	AKD
AKS	AKV	AKS
AKV	AKV	AKV

Pragmatics of PE

```
int{42} main() {  
    int{21} a;  
    int{2} b;  
    int{42} res;  
  
    a = 21;  
    b = 2;  
    res = a * b;  
    return res;  
}
```




CF (Constant Folding):

```
For each expr E do {  
    if( type(E) ==  $\alpha$ {val} ) {  
        replace E by val;  
    }  
}
```

More PE

SCCF (Structural-Constant Constant Folding)

```
{  
  .....  
  x = [a,b,c,d,e,f];  
  .....  
  ..... X[[2]] .....  
}
```



```
{  
  .....  
  x = [a,b,c,d,e,f];  
  .....  
  ..... C .....  
}
```

```
[a,b,c,] ++ [d,e]
```



```
[a,b,c,d,e]
```

.....



And More

SCS (Symbolic Constant Simplification)

$\max(X, X) \rightarrow X$

$X == X \rightarrow \text{true}$

.....

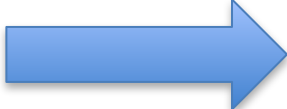


And More....

SCWLF (Simple Composition With-Loop Folding)



```
{
  ....
  a = with {
    ( [1,1] <= iv < [3,3] ) : f(iv);
  } : genarray( [5,8], def);
  ....
  return a[ [2,2] ];
}
```



```
{
  ....
  return f([2,2]);
}
```

And More....

WLF (With-Loop Folding)



```
{
  ....
  a = with {
    ( [1,1] <= iv < [3,3] ) : f(iv);
  } : genarray( [5,8], def);

  b = with {
    ( [1,1] <= iv < [2,2] ) : a[iv];
  } : genarray( [5,8], def);
  return b;
}
```

→

```
{
  ....
  b = with {
    ( [1,1] <= iv < [2,2] ) : f(iv);
  } : genarray( [5,8], def);
  return b;
}
```


And More And More And More And More....

Hold on!

What, exactly, is PE ????

We execute redices at compile time if they are

- δ -redices, or
- β -redices with no more than one occurrence of the argument in the function body

Also referred to as Supercompilaton



Intermission

Church-Rosser
Property...

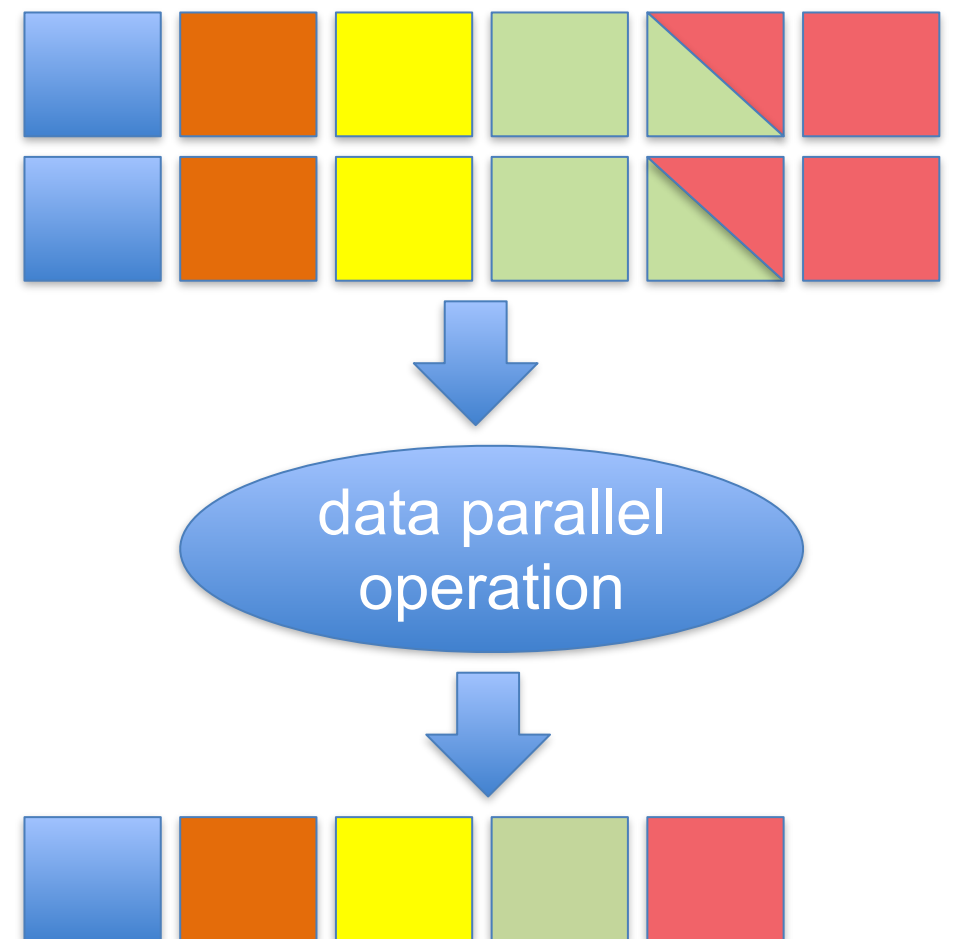
Is the kind of optimisation we do actually legit?

strict semantics...

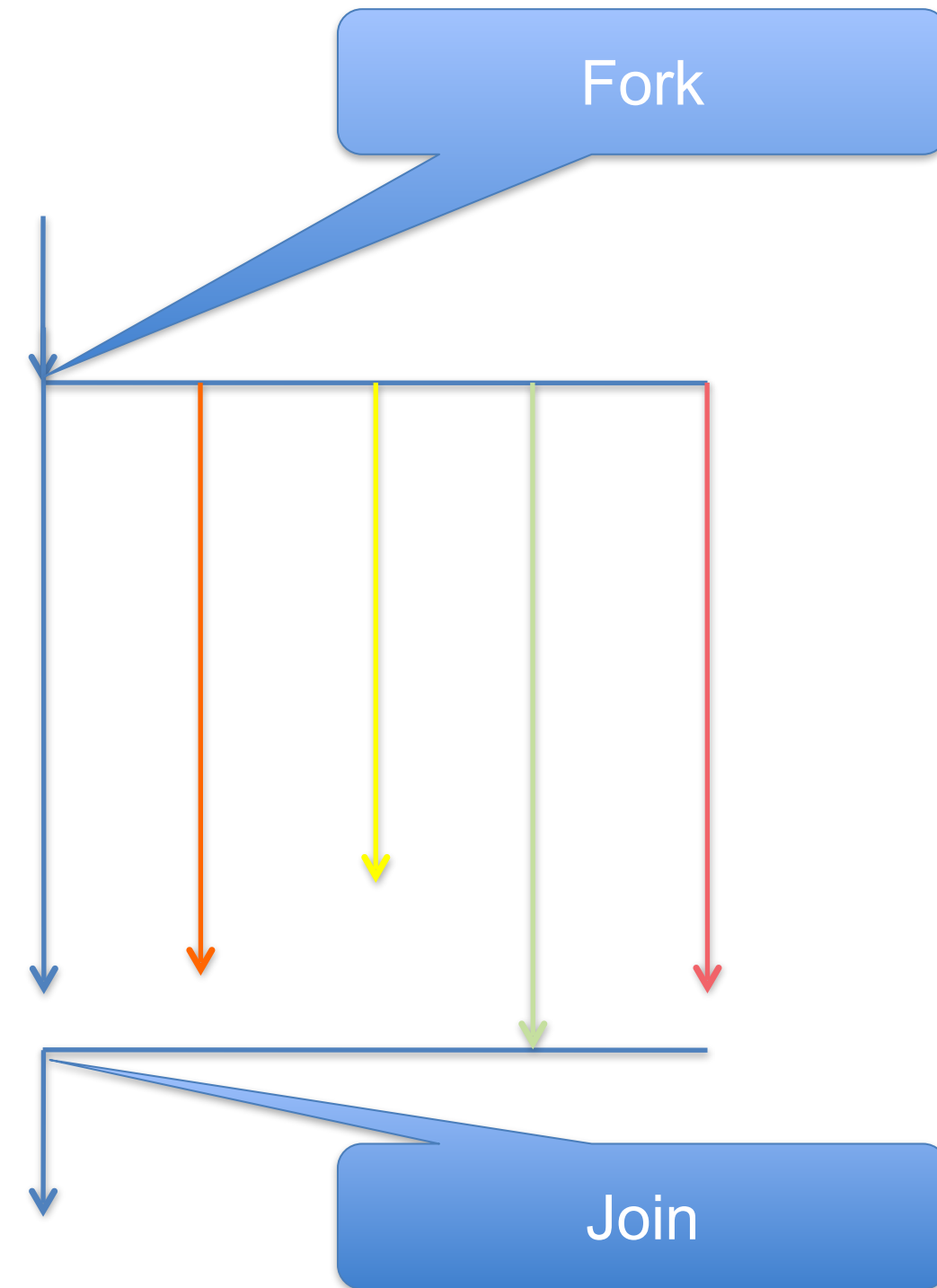
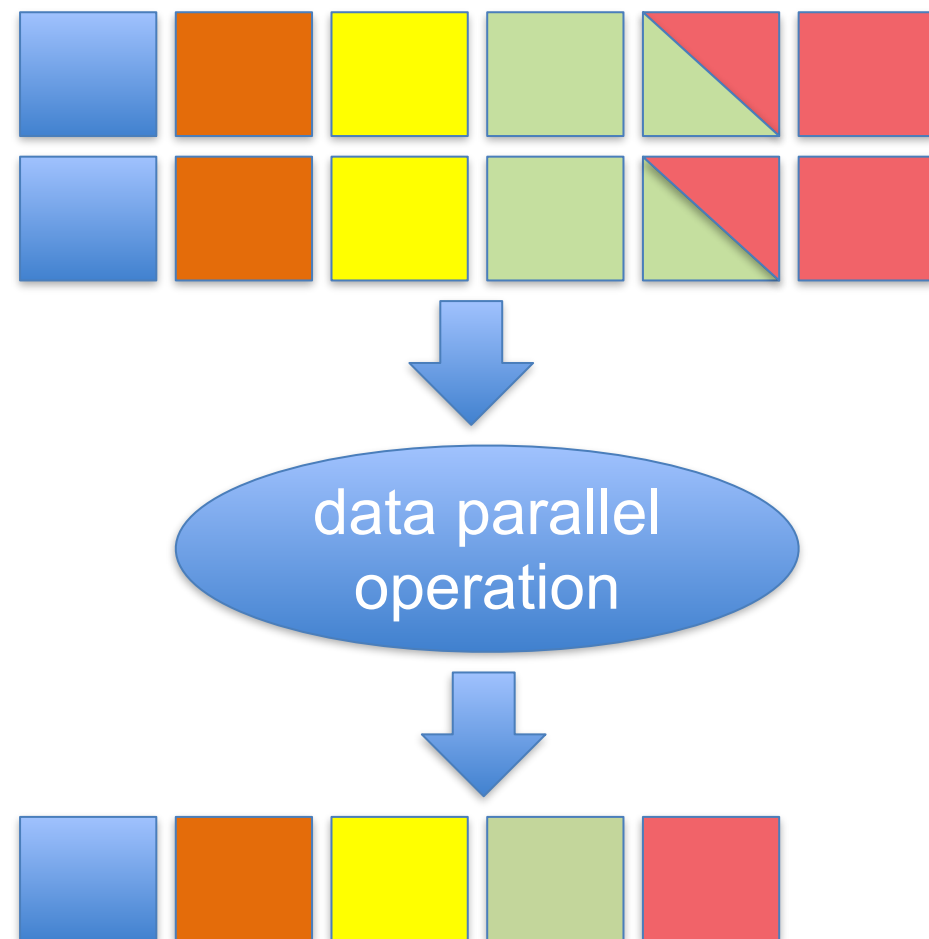
Where does Parallelism Enter the Game?

```
res = with {  
  ([0] <= [i] <[3]) : f(a[0,i],    a[1,i]);  
  ([3] <= [i] <[5]) : g(a[0,i],    a[1,i],  
                        a[0,i+1], a[1,i+1]);  
} : genarray ([5], 0)
```

All combinators are defined
through With-Loops!



The Fork-Join Model of With-Loops / Tensor Comprehensions



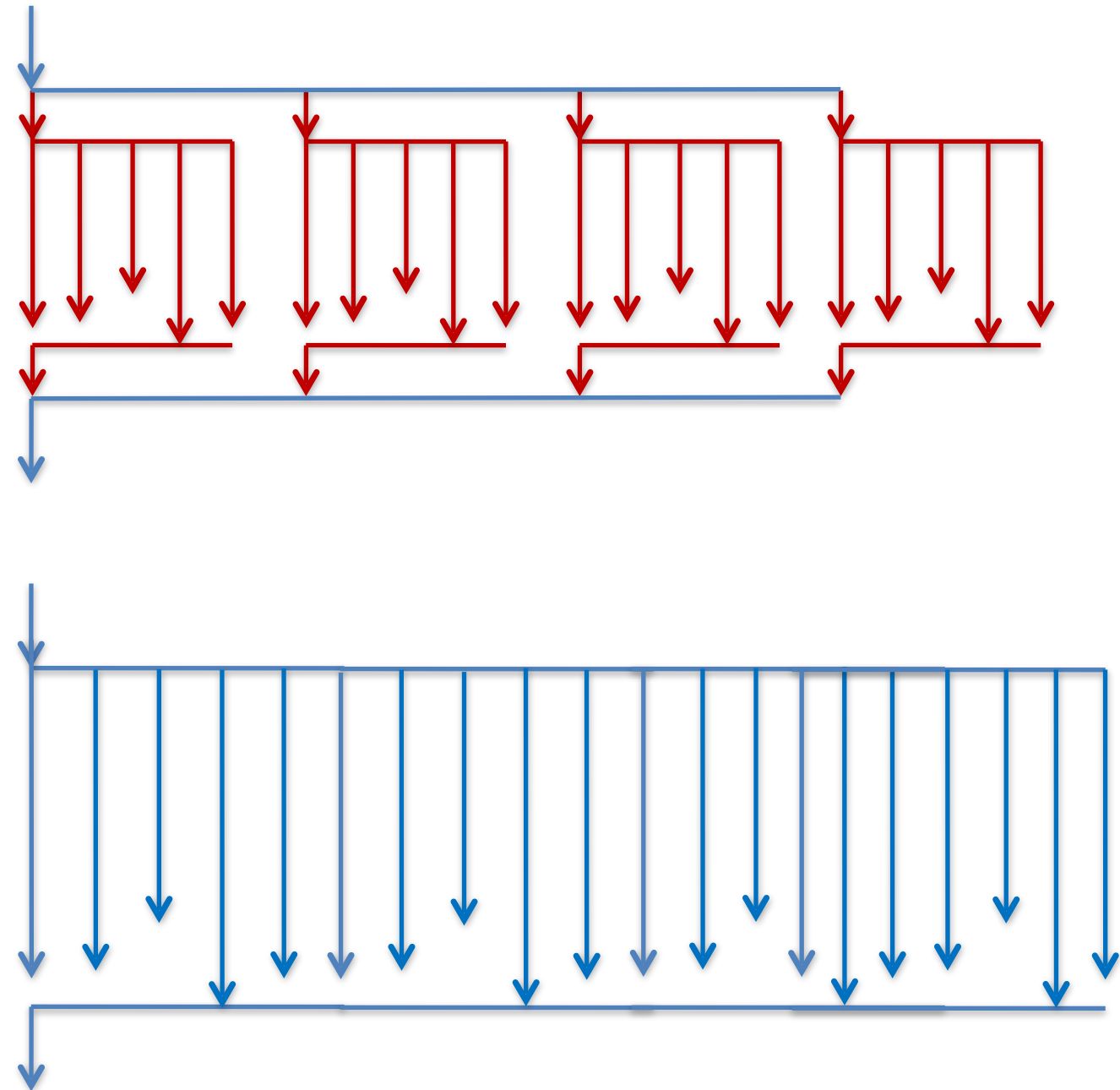
Nested With-Loops

$\{ [i] \rightarrow \{ [j] \rightarrow a[i,j] + b[i,j] \} \}$



With-Loop-Scalarization

$\{ [i,j] \rightarrow a[i,j] + b[i,j] \}$



Nested With-Loops

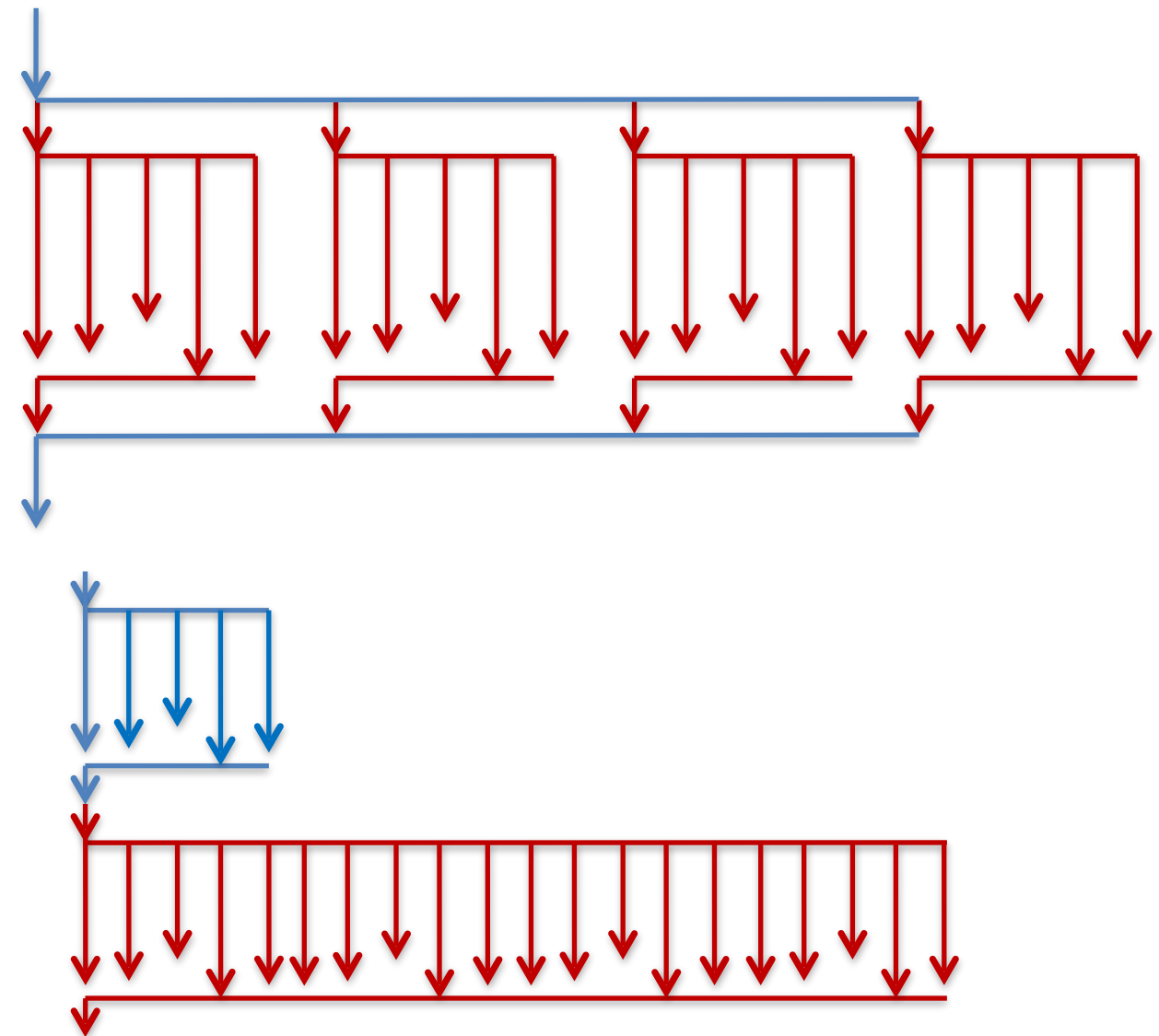
$\{ [i] \rightarrow \text{foo}(i) + \{ [j] \rightarrow a[i,j] + b[i,j] \} \}$

With-Loop-Scalarization ???

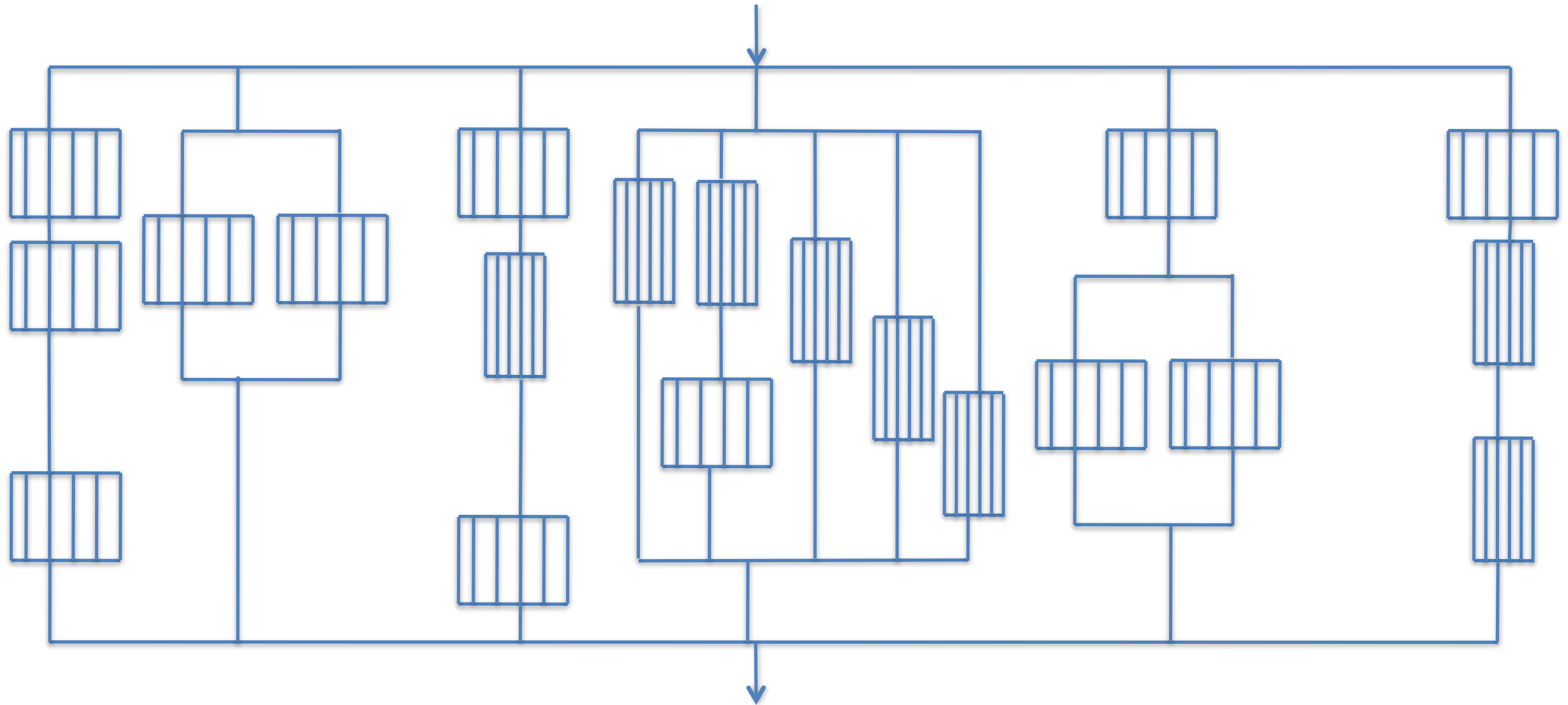
$\{ [i,j] \rightarrow \text{foo}(i) + a[i,j] + b[i,j] \}$

With-Loop-Lifting ???

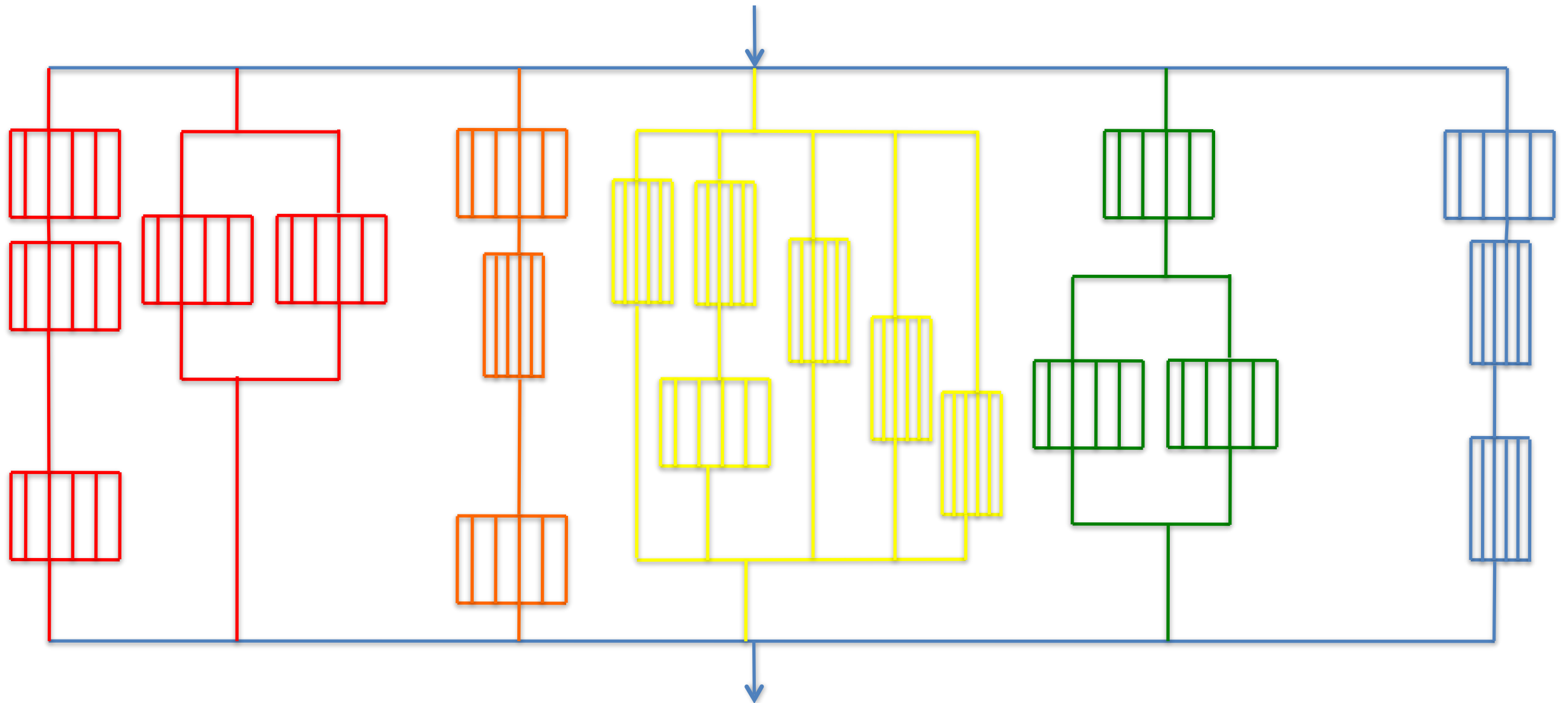
$\text{tmp} = \{ [i] \rightarrow \text{foo}(i) \};$
 $\{ [i,j] \rightarrow \text{tmp}[i] + a[i,j] + b[i,j] \}$



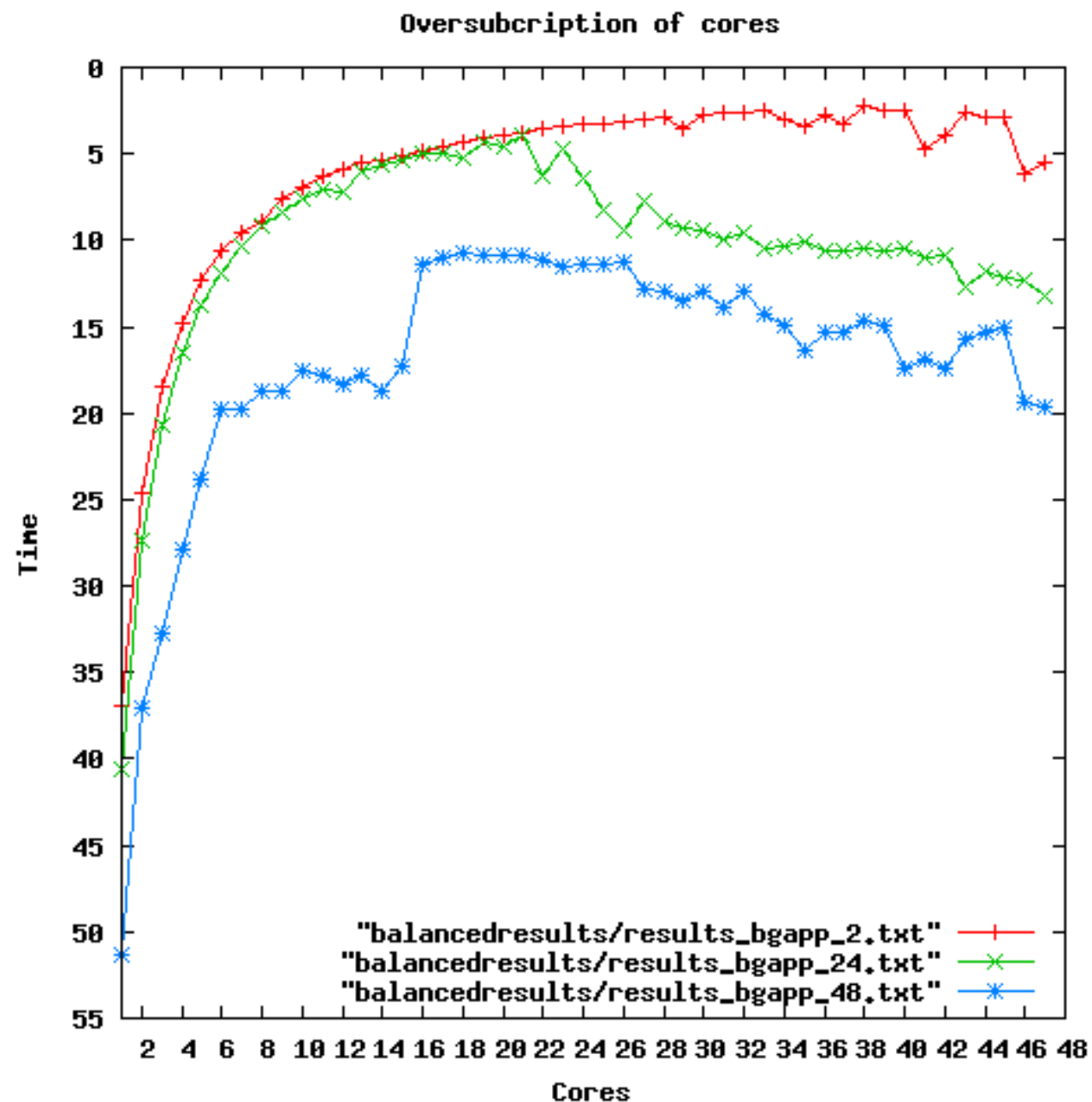
Looking at Entire Programs:



SMP Ideal: Coarse Grain!



The Need for Dynamic Adaptation



[GS15] Stuart Gordon, Sven-Bodo Scholz, Dynamic Adaptation of Functional Runtime Systems Through External Control Selected Papers IFL'14, . pp. 10:1–10:13. ACM. New York, NY, USA.

Array-Layout Manipulation for Improved Vectorisation

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42

Shape: [6,7]

Layout "2"

Shape: [6,2,4]

				29	30	31	32
		22	23	24	25		
	15	16	17	18			
8	9	10	11				
1	2	3	4				
5	6	7					

Layout "1"

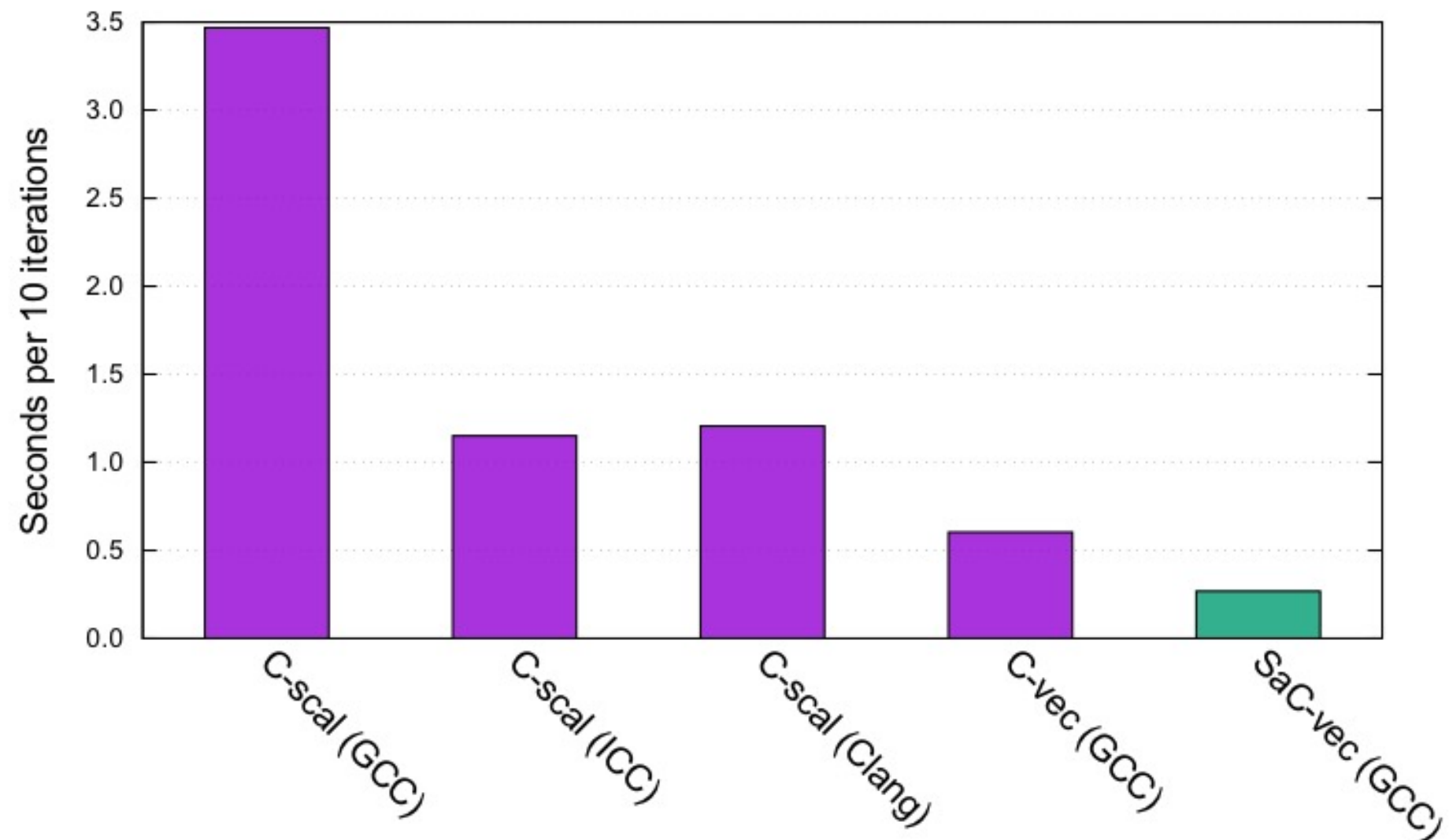
Shape: [2,7,4]

		29	36			
		30	37			
1	8	15	22			
2	9	16	23			
3	10	17	24			
4	11	18	25			
5	12	19	26			
6	13	20	27			
7	14	21	28			

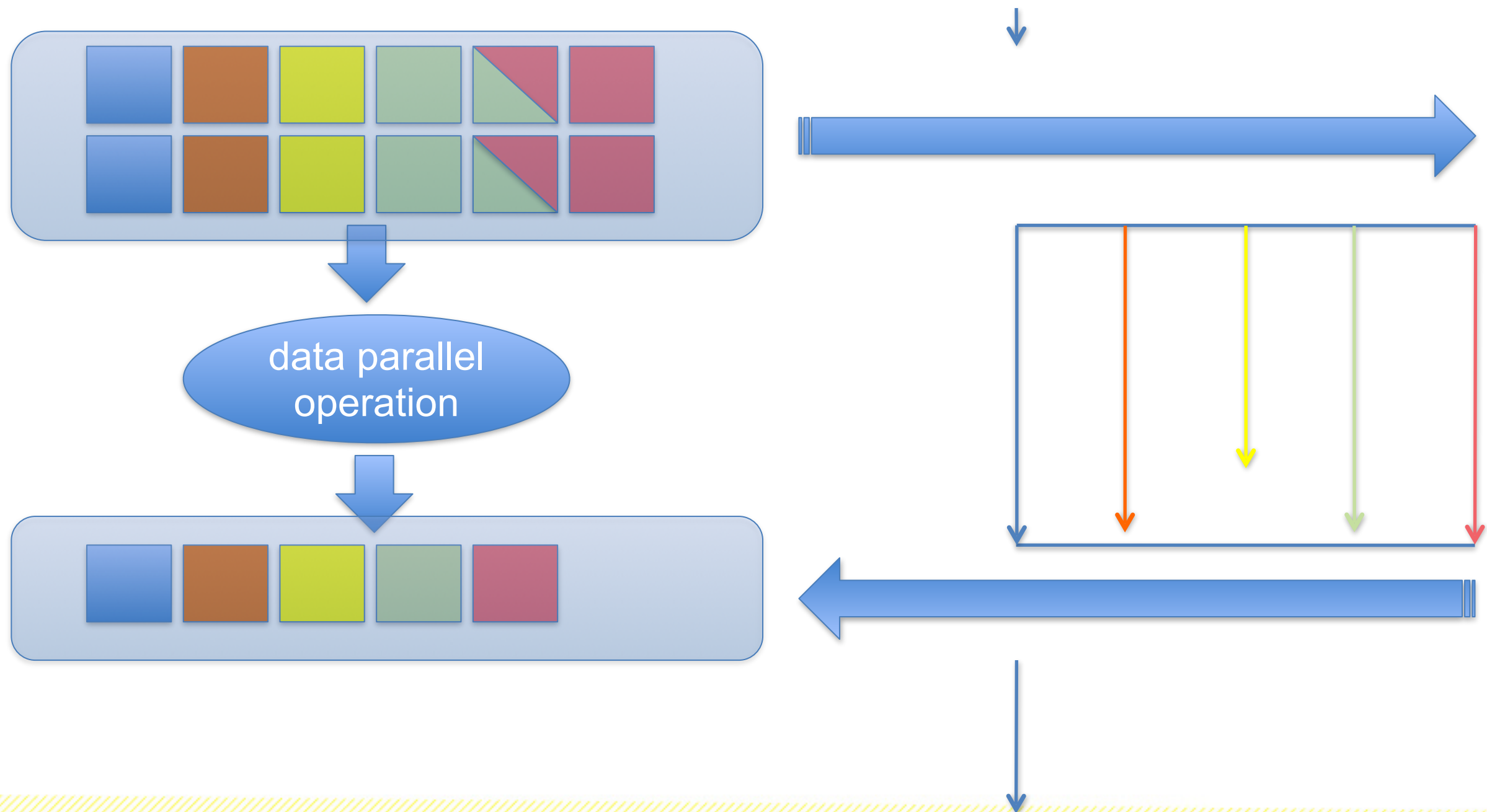
[SS15] A. Šinkarovs and S.-B. Scholz, Type-driven data layouts for improved vectorisation, Wiley CCPE 2015.

Impact of the Type-Driven Data Layout Transformation

Spectral Norm on 2000x2000 elements, single core i3

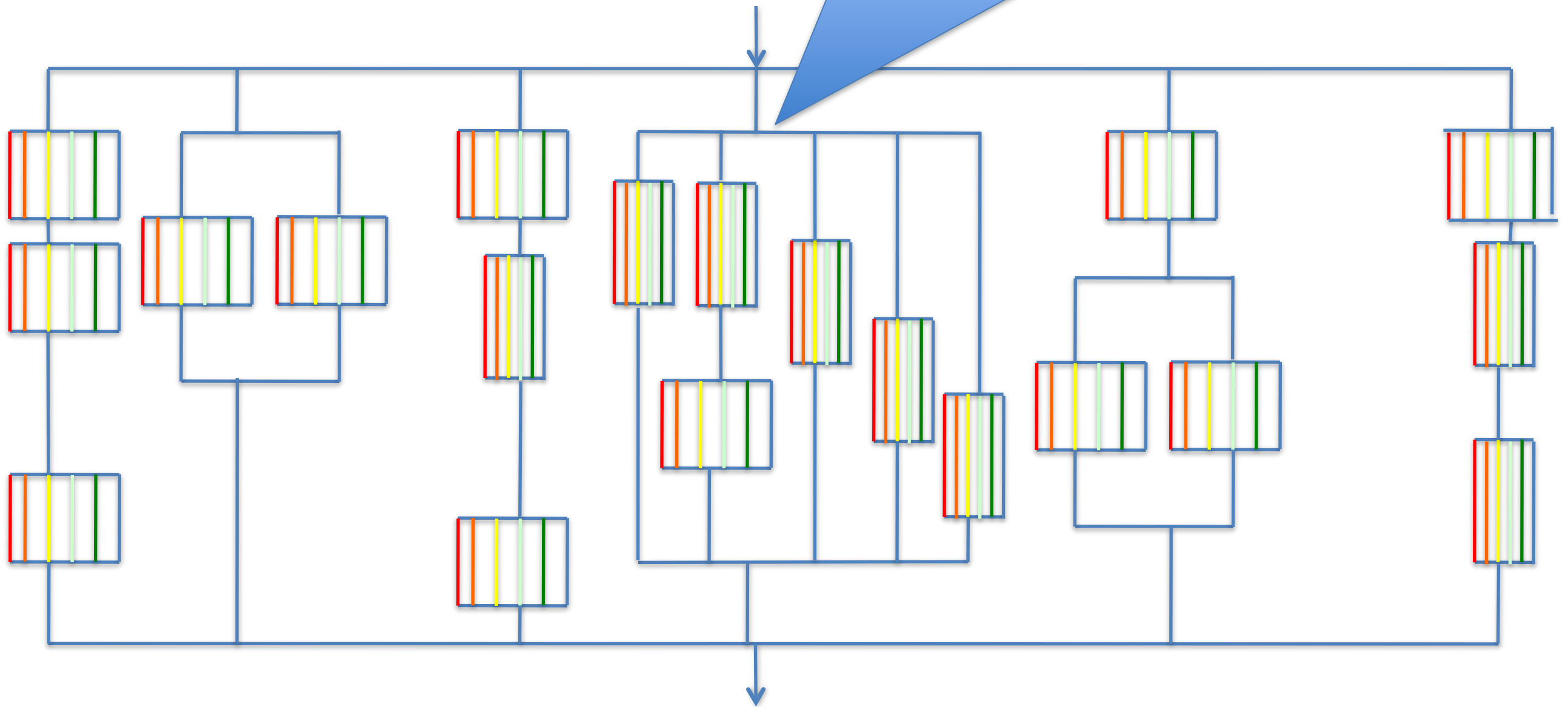


Multi-Threaded Execution on GPGPUs

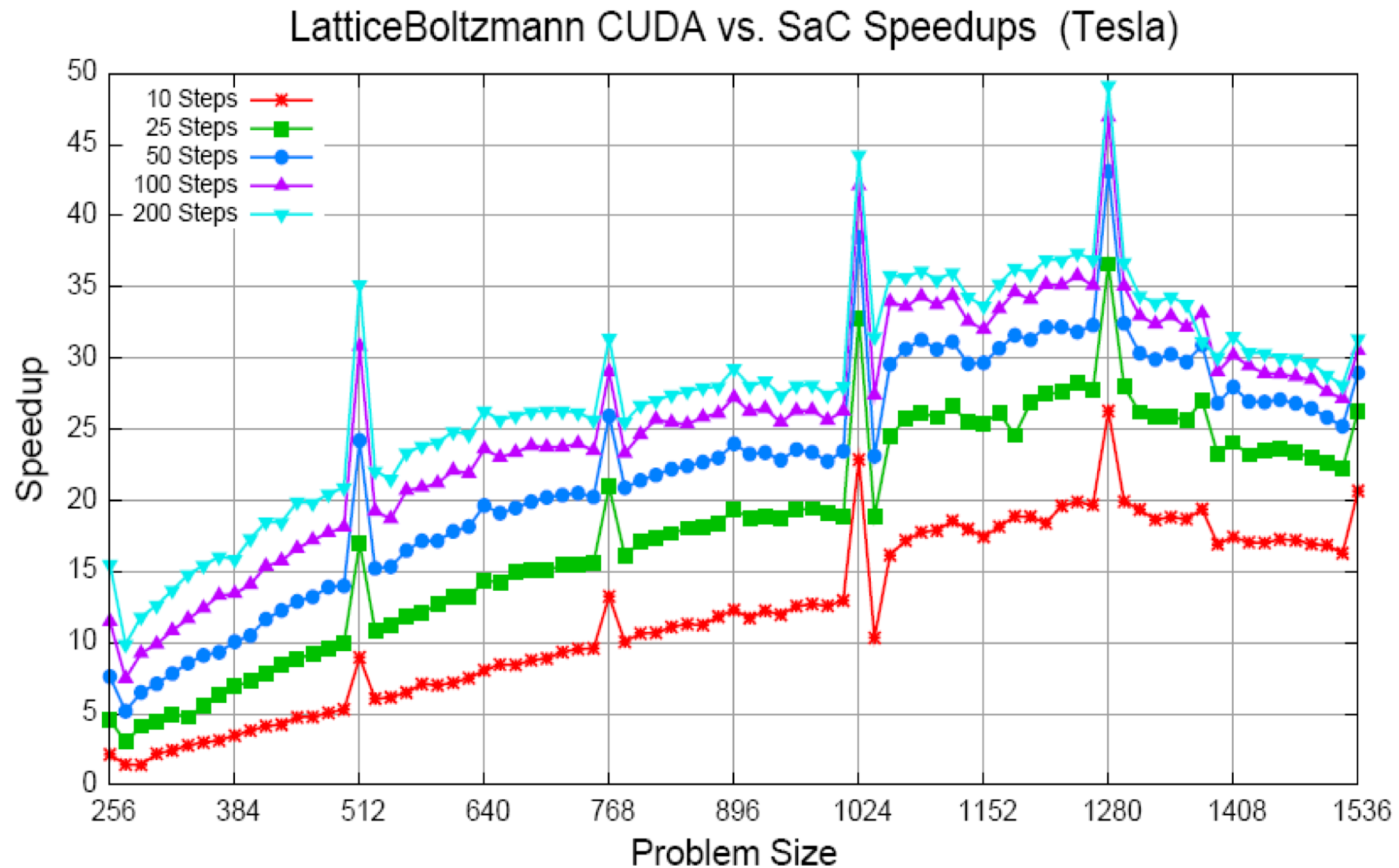


GPGPU Ideal: Homogeneous Flat Coarse Grain

NB: This choice is different from SMP!!



GPUs and synchronous memory transfers



GPGPU Naive Compilation

```
PX = { [i,j] -> PX[ [i,j] ] + sum( VY[ [.,i] ] * CX[ [i,j] ] ) };
```



```
PX_d = host2dev( PX);  
CX_d = host2dev( CX);  
VY_d = host2dev( VY);
```

```
PX_d = { [i,j] -> PX_d[ [i,j] ] + sum( VY_d[ [.,i] ] * CX_d[ [i,j] ] ) };
```

```
PX = dev2host( PX_d);
```

CUDA
kernel

Hoisting memory transfers out of loops

```
for( i = 1; i < rep; i++) {
```

```
    PX_d = host2dev( PX);  
    CX_d = host2dev( CX);  
    VY_d = host2dev( VY);
```

```
    PX_d = { [i,j] -> PX_d[ [i,j] ] + sum( VY_d[ [.,i] ] * CX_d[ [i,j] ] ) };
```

```
    PX = dev2host( PX_d);
```

```
}
```

Retaining Arrays on the Device

```
CX_d = { [i,j] -> ..... } ;
```

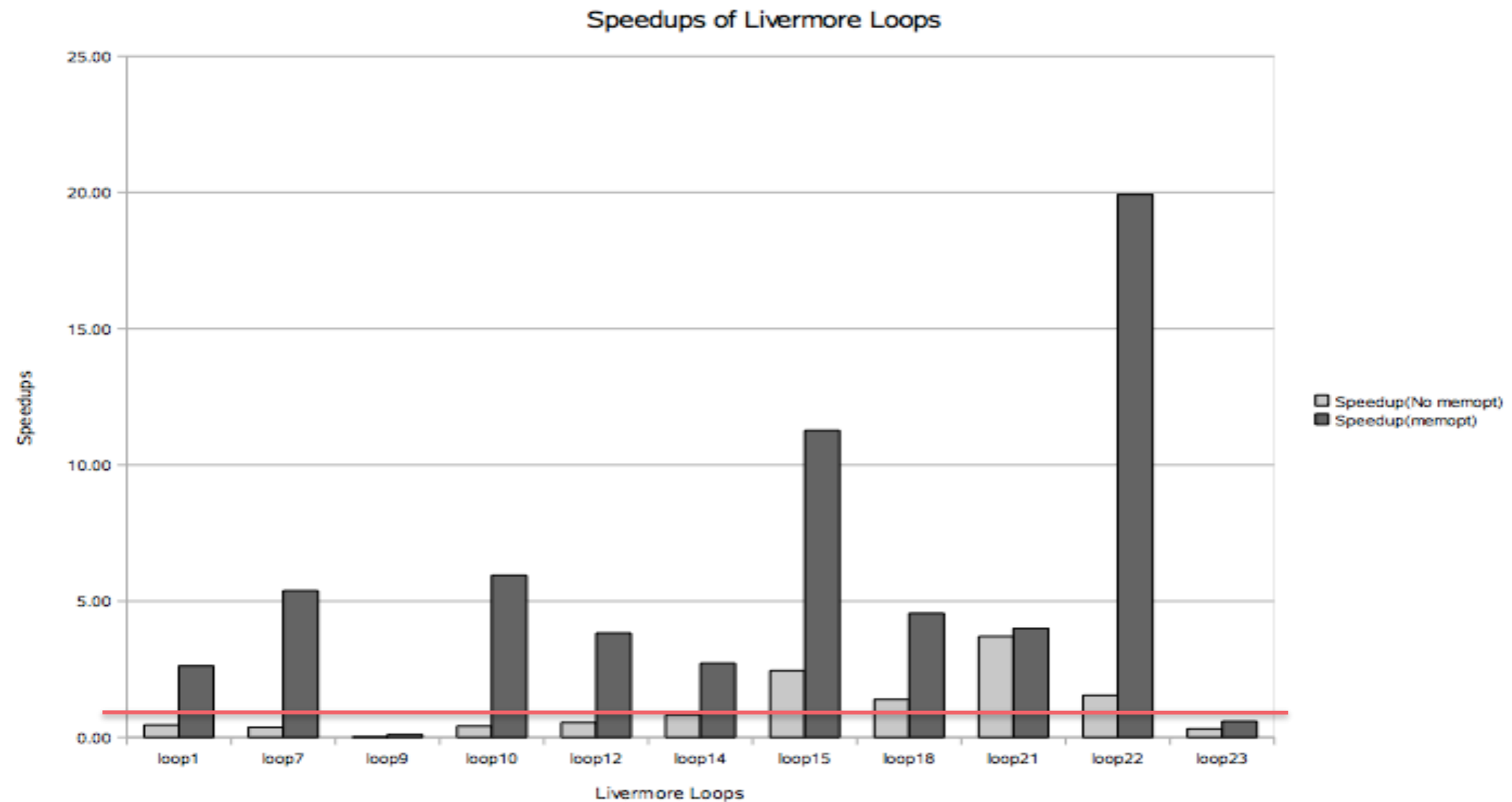
```
CX = dev2host( CX_d);
```

```
PX_d = host2dev( PX);
```

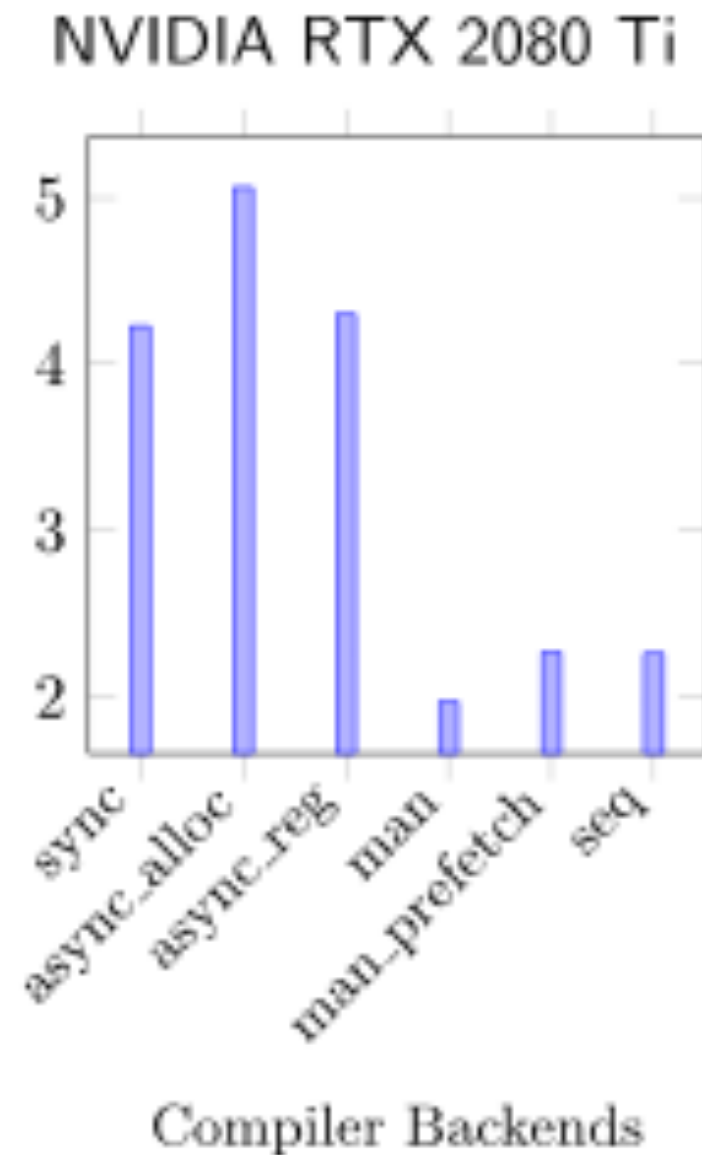
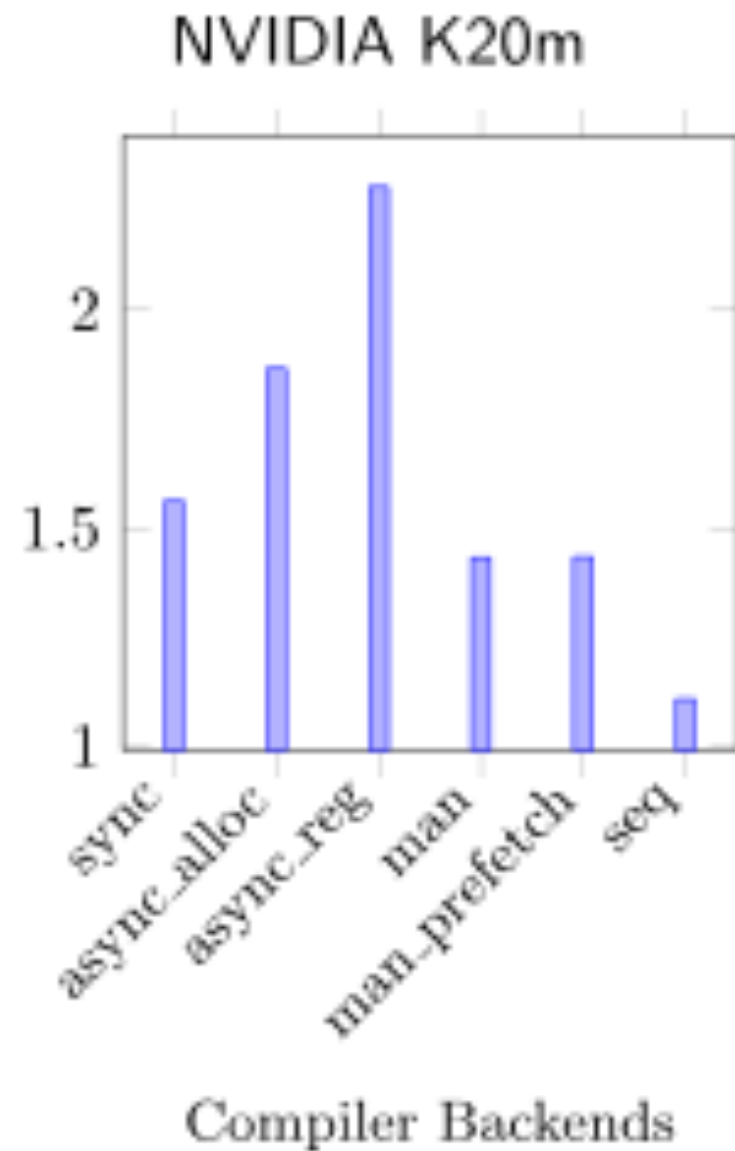
```
VY_d = host2dev( VY);
```

```
for( i = 1; i< rep; i++) {
```


The Impact of Redundant Memory Transfers



Effects of using different GPU memory models



[VS20] H. Viessmann
and S.-B. Scholz,
Effective Host-GPU
Memory Management
Through Code
Generation, to appear.

SaC Tool Chain

- `sac2c` – main compiler for generating executables; try
 - `sac2c -h`
 - `sac2c -o hello_world hello_world.sac`
 - `sac2c -t mt_pth`
 - `sac2c -t cuda`
- `sac4c` – creates C and Fortran libraries from SaC libraries
- `sac2tex` – creates TeX docu from SaC files

Outlook

- There are still many challenges ahead, e.g.
 - Higher-Order functions / defunctionalisation
 - Element-Type polymorphism / monomorphysation
 - Joining data and task parallelism
 - Better memory management
 - Application studies
 - Novel Architectures
 - ... and many more ...
- If you are interested in joining the team / doing an internship or a master project:
 - talk to me 😊