Polyhedral Methods for Improving Parallel Update-in-Place

IMPACT'14

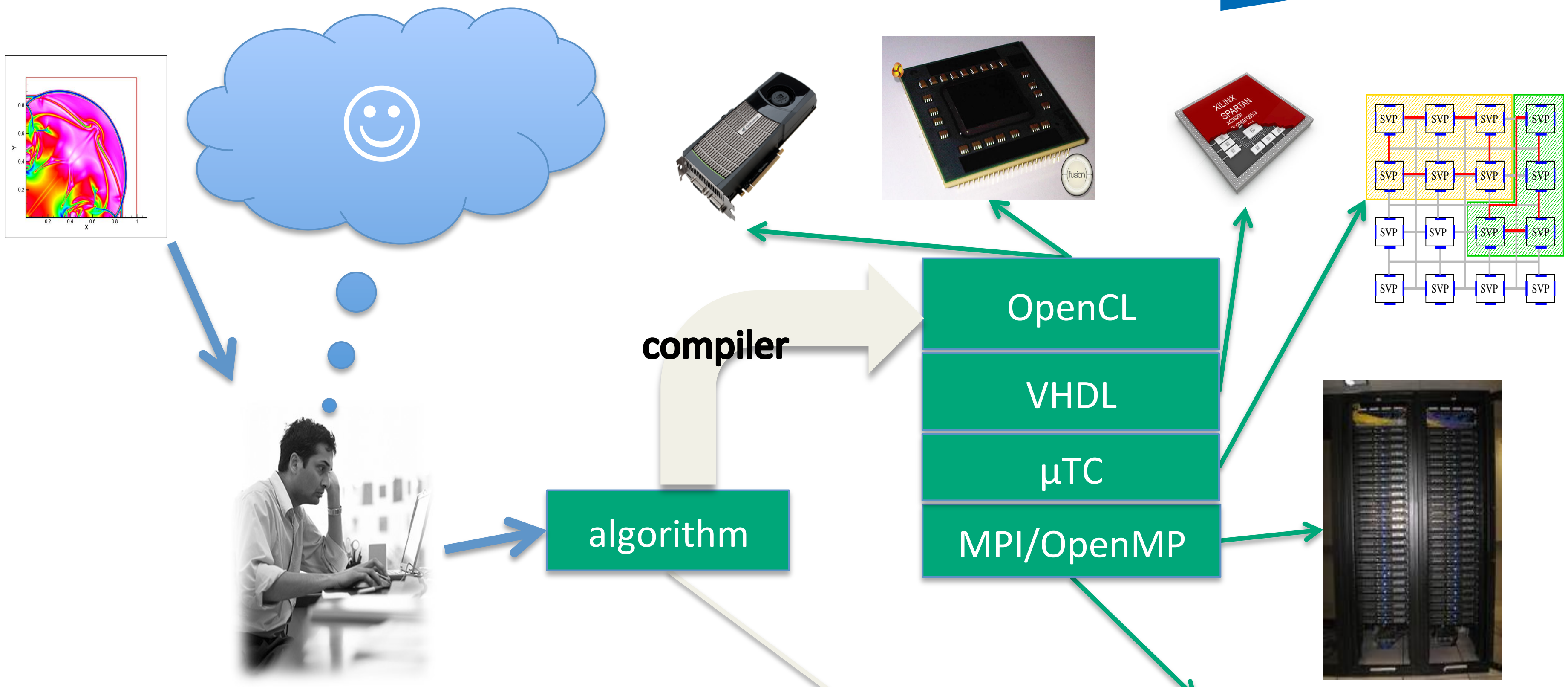
January 20, 2014, Vienna

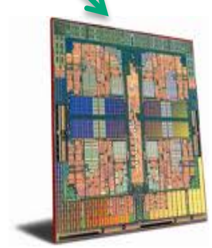
J. Guo, R. Bernecky, Jeyan T., S-B. Scholz

Clemens Grelck

(University of Hertfordshire, Snake Island Research, Oxford e-Research Centre, Heriot-Watt University, University of Amsterdam)

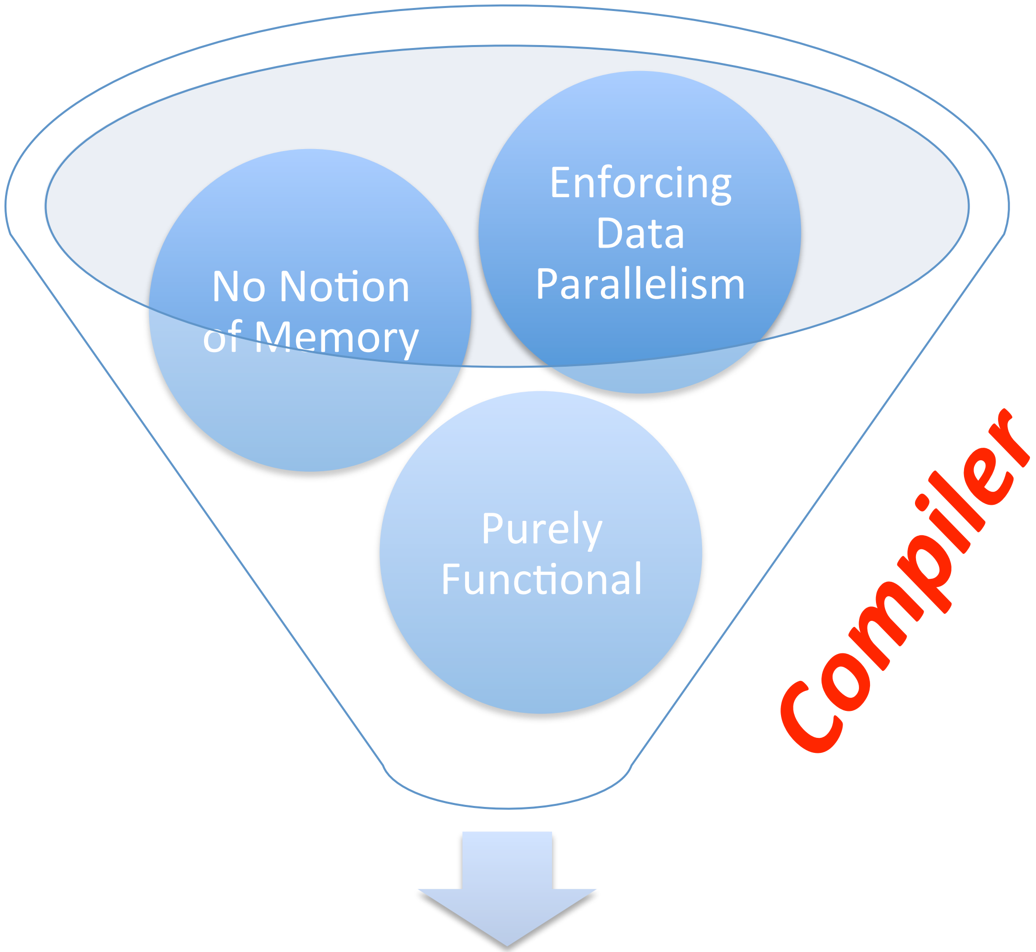
HP³ Vision





High **P**roductivity
High **P**erformance
High **P**ortability

“Magic” Ingredients



HP^3

With-Loops in SaC

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

def	def	def	def	def	def	def	def
def	**f([1,1])**	**f([1,2])**	**f([1,3])**	def	def	def	def
def	**f([2,1])**	**f([2,2])**	**f([2,3])**	def	def	def	def
def	**f([3,1])**	**f([3,2])**	**f([3,3])**	def	def	def	def
def	def	def	def	def	def	def	def

Memory Management

allocate

Iterate & fill

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


b = … a ………… a ……..


c = ………. a ……..
```

free

Memory Reuse Example

```
b = with {
        ( [1,1] <= iv < [3,3] ) : f(iv);
    } : modarray( a);
```

a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	**f([1,1])**	**f([1,2])**	**f([1,3])**	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	**f([2,1])**	**f([2,2])**	**f([2,3])**	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	**f([3,1])**	**f([3,2])**	**f([3,3])**	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]

Can we reuse existing array a to represent new array b ?

- Smaller memory footprint
- Avoid copying overhead
- Depends on reference count

Memory Reuse ?

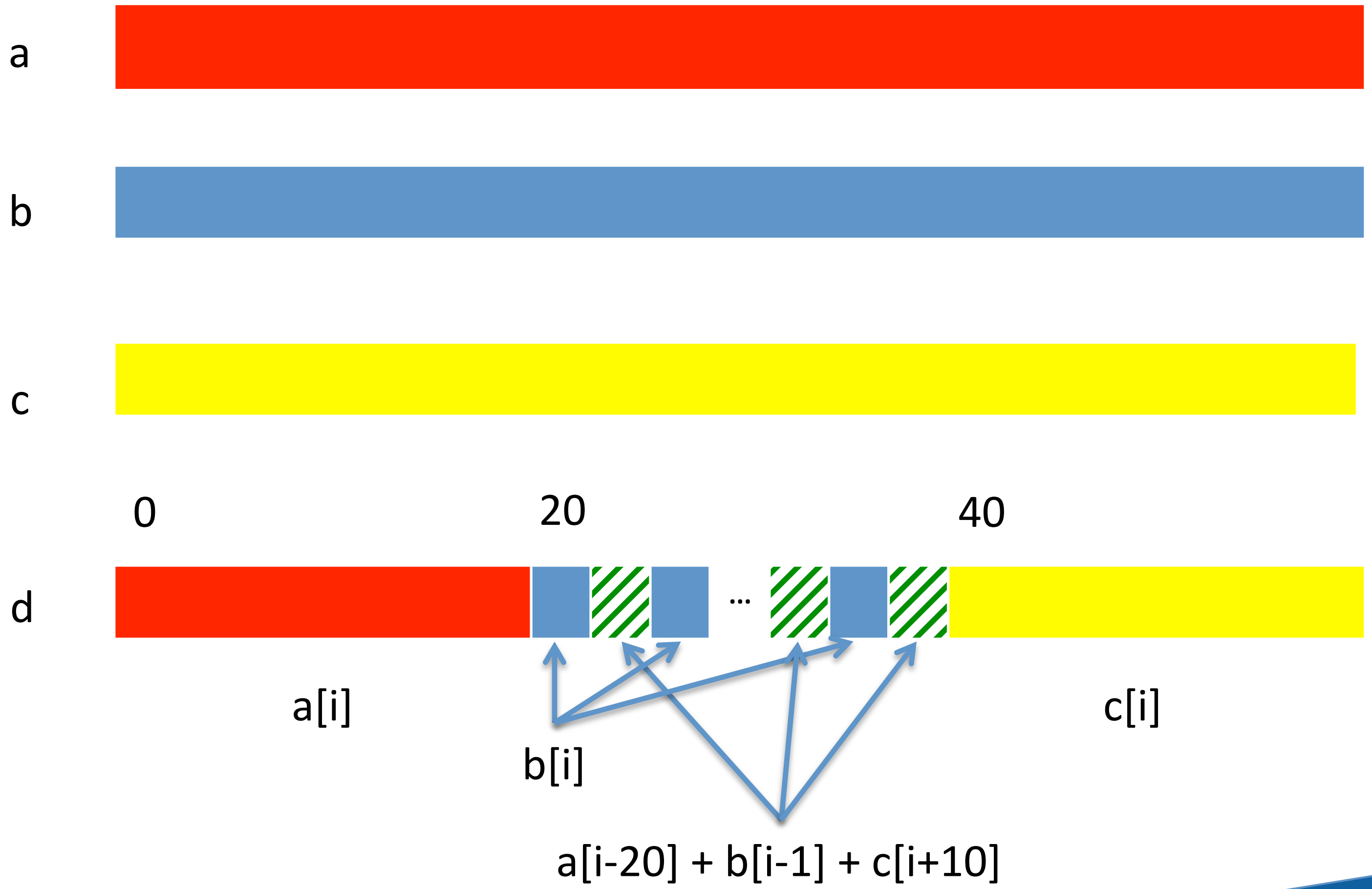
a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	**f([1,1])**	**f([1,2])**	**f([1,3])**	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	**f([2,1])**	**f([2,2])**	**f([2,3])**	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	**f([3,1])**	**f([3,2])**	**f([3,3])**	a[iv]	a[iv]	a[iv]	a[iv]
a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]	a[iv]

Can we recycle* a *?

f(iv) = 42 ✓
f(iv) = c[iv+1] ✓
f(iv) = a[iv] ✓
f(iv) = a[iv + [1,0]] 🚫
f(iv) = a[iv + [3,0]] ✓
f(iv) = a[0,0] ✓

General Case

Which array (if any) can we recycle?



Observations

Polyhedral representations lend themselves!

Polyhedral tooling may be used!

Identifying *R*ead *D*ata *S*paces

foreach *Array access* $\mathcal{A}[\mathcal{I}v]$ *in* $\mathcal{W}l$ **do**

  Let $\mathcal{I}$ be the iteration vector associated with this access;

  Let **CP** be the set of control paths associated with this access;

  if $\mathcal{A}[\mathcal{I}v]$ *is a non-in-place read access* **then**

    $\mathcal{A}ffine$, $\mathcal{DS} \leftarrow$ `Get_Data_Space(`$\mathcal{I}$, $\mathcal{I}v$, **CP**, $\mathcal{SC}iv$`);`

    if $\mathcal{A}ffine = True$ **then**

      $\mathbf{RDS}[\mathcal{A}] \leftarrow \mathbf{RDS}[\mathcal{A}] \cup \mathcal{DS}$;

    else

      Terminate;

Identifying Copy Data Spaces

else if $\mathcal{A}[\mathcal{I}v]$ *is the global write access in a copy assignment* **then**

  Let $\mathcal{A}'[\mathcal{I}v]$ be the in-place read of this copy assignment;

  $\mathcal{A}ffine, \mathcal{DS} \leftarrow$ `Get_Data_Space(`$\mathcal{I}$, $\mathcal{I}v$, $\mathbf{CP}$, $\mathcal{SC}iv$`)`;

  if $\mathcal{A}ffine = True$ **then**

    $\mathbf{CDS}[\mathcal{A}'] \leftarrow \mathbf{CDS}[\mathcal{A}'] \cup \mathcal{DS}$;

  else

    Terminate;

else

  Continue;

Identifying *Reuse* *Candidates*

$\mathbf{RC} \leftarrow \emptyset;$

$\mathbf{foreach}\ \mathcal{A}\ \mathit{with}\ (\mathcal{A} \mapsto \mathcal{CDS}) \in \mathbf{CDS}\ \mathbf{do}$

$\quad \mathbf{if}\ \mathbf{RDS}[\mathcal{A}] \subseteq \mathbf{CDS}[\mathcal{A}]\ \mathbf{then}$

$\quad\quad \mathbf{RC} \leftarrow \mathbf{RC} \cup \{\mathcal{A}\};$

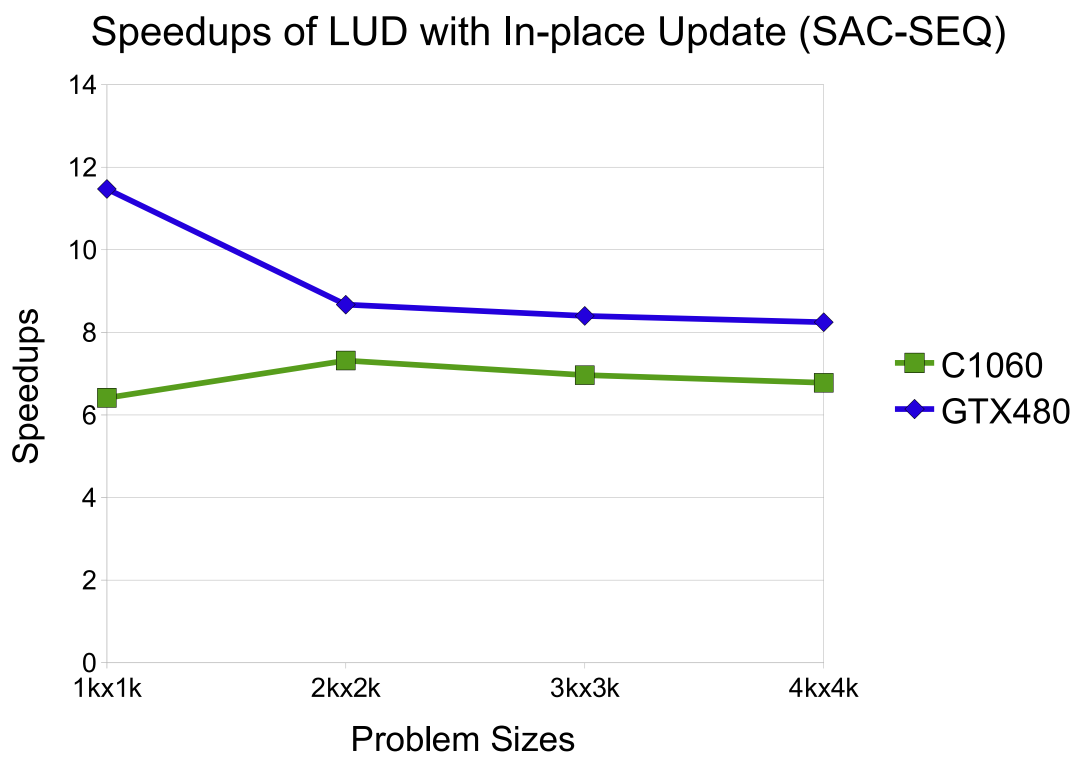
Use the PolyLib for polyhedral operations:

- Union of data spaces
- Inclusion relationship
- Image of polytope under affine tranformation

Example: LU Decomposition

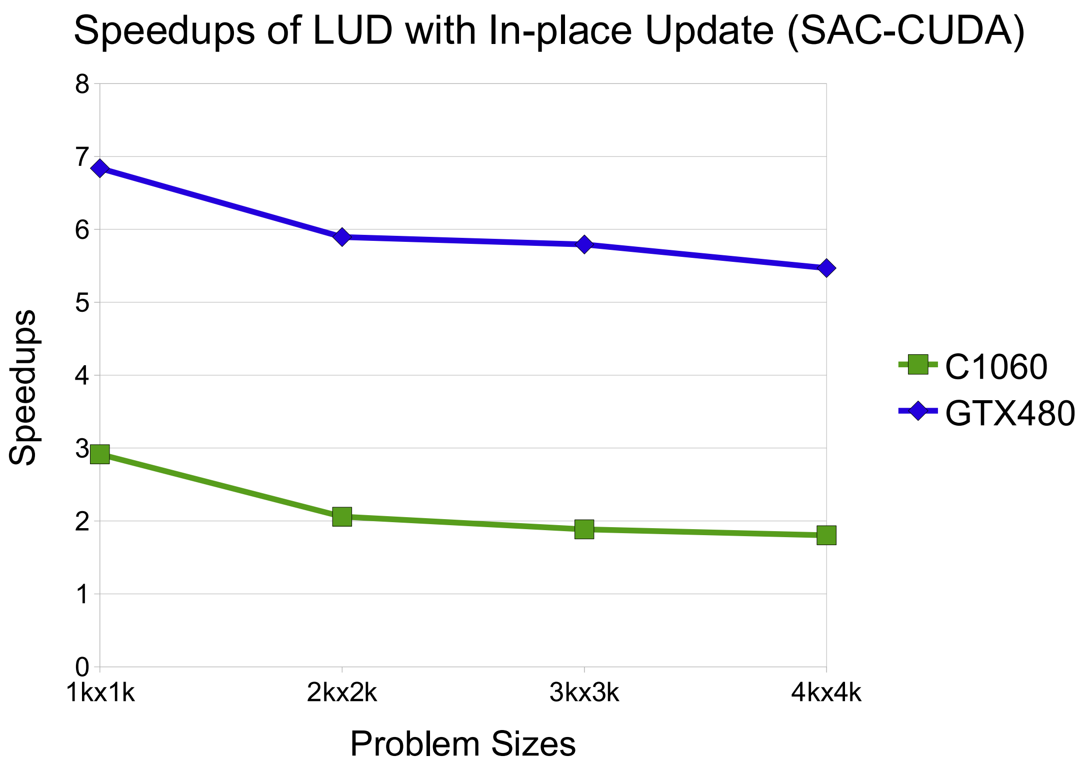
```
for( k = 0; k < N-1; k++) {
  A = with {
        ( [k+1,k] <= [i,j] < [N,k+1])
                                        : A[i,j]/A[k,k];
      } : modarray( A);
  A = with {
        ( [k+1,k+1] <= [i,j] < [N,N])
                                  : A[i,j]-A[i,k]*A[k,j];
      } : modarray( A);
}
```

Runtime Impact (single core)



	-noPRA	-doPRA
1024 x 1024	0.28 GFLOPS	1.49 GFLOPS
2048 x 2048	0.19 GFLOPS	1.33 GFLOPS
4096 x 4096	0.19 GFLOPS	1.29 GFLOPS

Runtime Impact (GPU)



	-noPRA	-doPRA
1024 x 1024	0.56 GFLOPS	10.21 GFLOPS
2048 x 2048	0.97 GFLOPS	11.22 GFLOPS
4096 x 4096	1.15 GFLOPS	11.39 GFLOPS

Memory Impact

	-noPRA	-doPRA
1024 x 1024	16 MB	9 MB
2048 x 2048	66 MB	34 MB
4096 x 4096	258 MB	138 MB

Conclusions

- Polyhedral model is an excellent means to formalise array accesses
- Even a dependency free setting benefits from the full power
- Reuse enables copy elimination
- Leads to essential speedups/ bridges the gap to classical for loops