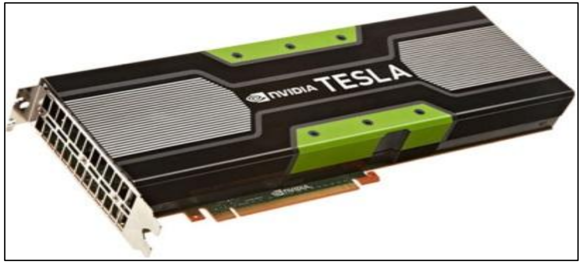
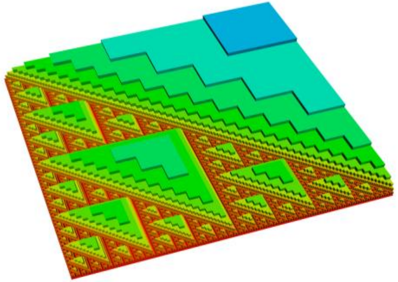
Application Specific Computing (ASC)

Robert Strzodka
Application Specific Computing
ZITI, Heidelberg University
http://asc.ziti.uni-heidelberg.de

Application Specific Computing (ASC)

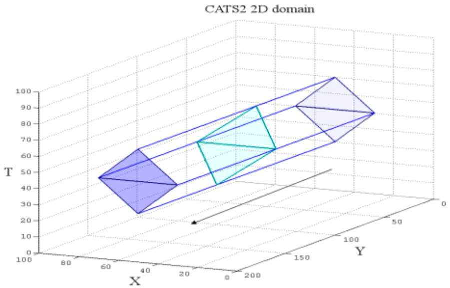
Accelerator





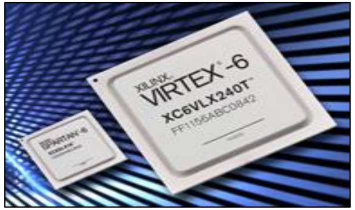
Data

Many-core

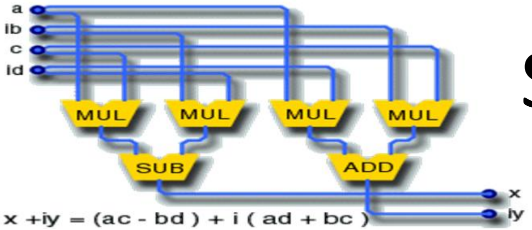


Algorithm

FPGAs



Embedded



Software

2

3

#	T	Site	Manufacturer	Computer	Country	HPCG [Pflop/s]	Rmax [Pflop/s]	HPCG/ Peak	HPCG/ HPL
1	1	Oak Ridge National Laboratory	IBM	Summit IBM Power System, P9 22C 3.07 GHz, Volta GV100, EDR	USA	2.9258	122.3	1.6%	2.4%
2	3	Lawrence Livermore National Laboratory	IBM	Sierra IBM Power System, P9 22C 3.1 GHz, Volta GV100, EDR	USA	1.7957	71.6	1.5%	2.5%
3	16	RIKEN Advanced Institute for Computational Science	Fujitsu	K Computer SPARC64 VIIIfx 2.0GHz, Tofu Interconnect	Japan	0.6027	10.5	5.3%	5.7%
4	9	Los Alamos NL / Sandia NL	Cray	Trinity Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries	USA	0.5461	14.1	1.2%	3.9%
5	6	Swiss National Supercomputing Centre (CSCS)	Cray	Piz Daint Cray XC50, Xeon E5 12C 2.6GHz, Aries, NVIDIA Tesla P100	Switzerland	0.4864	19.6	1.9%	2.5%
6	2	National Supercomputing Center in Wuxi	NRCPC	Sunway TaihuLight NRCPC Sunway SW26010, 260C 1.45GHz	China	0.4808	93.0	0.4%	0.5%
7	12	JCAHPC Joint Center for Advanced HPC	Fujitsu	Oakforest-PACS PRIMERGY CX1640 M1, Intel Xeons Phi 7250 68C 1.4 GHz, OmniPath	Japan	0.3855	13.6	1.5%	2.8%
8	10	Lawrence Berkeley National Laboratory	Cray	Cori Cray XC40, Intel Xeons Phi 7250 68C 1.4 GHz, Aries	USA	0.3554	14.0	1.3%	2.5%
9	14	Commissariat a l'Energie Atomique (CEA)	Bull	Tera-1000-2 Bull Sequana X1000, Intel Xeon Phi 7250 68C 1.4 GHz, Bull BXI 1.2	France	0.3338	12.0	1.4%	2.8%
10	8	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	0.3304	17.2	1.6%	1.9%

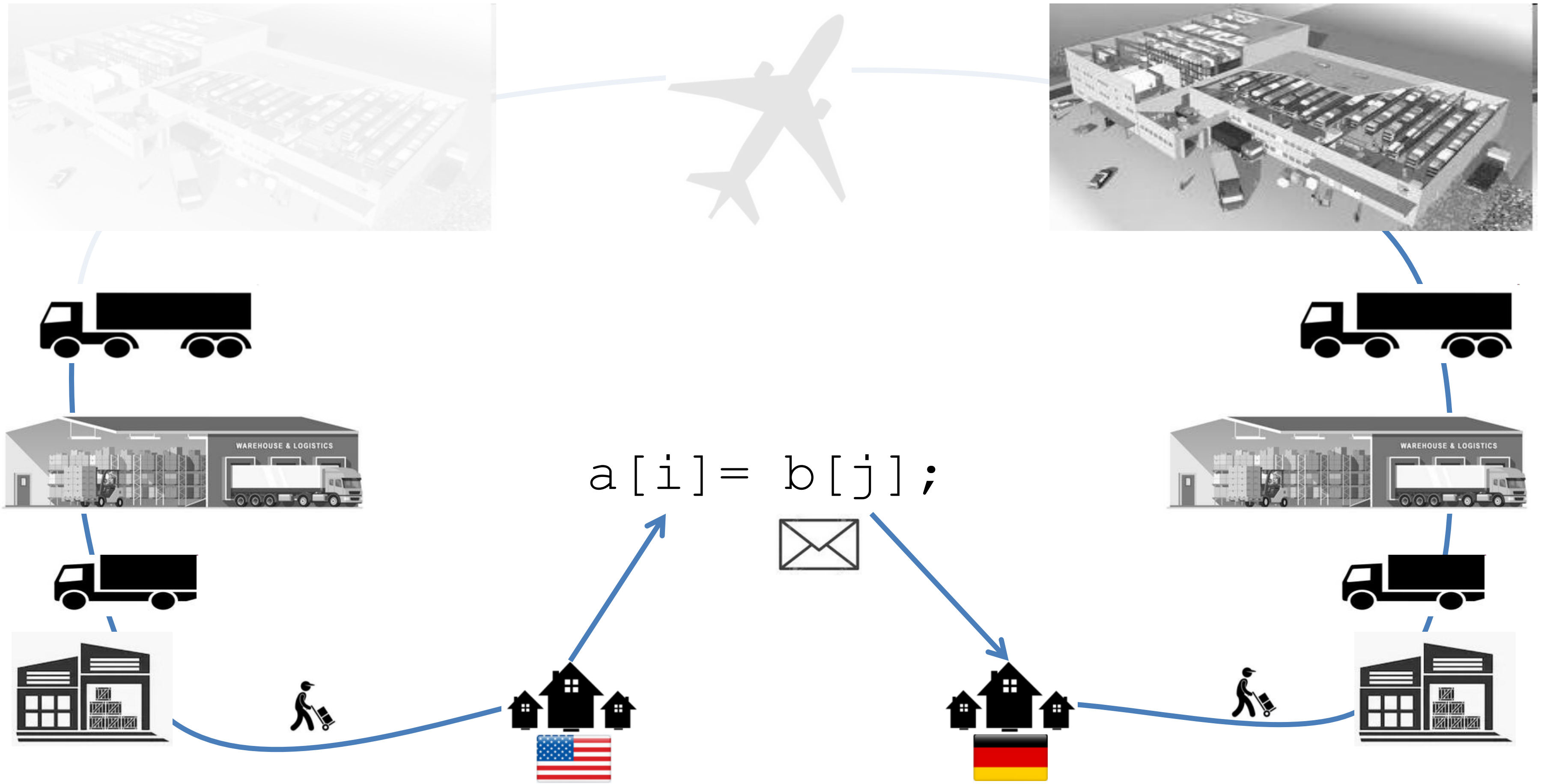
Source: www.top500.org

Why is Everything so Slow?

4

- Data: `a[i]= b[j]`
  - Number representation
  - Data ordering and layout
- Loops: `for(…)`
  - Tilings
  - Hierarchies
  - Fusion
- Branches: `if(…)`
  - Extraction
  - Pre-evaluation
  - Predication

Single Data Access as Postal Delivery



5

Loops, single access

7

Foreach and Tiling

```
for(i=0; i<N; ++i){
   S(i);
}
```

- Thread parallelism in outer loop
- SIMD parallelism in inner loop

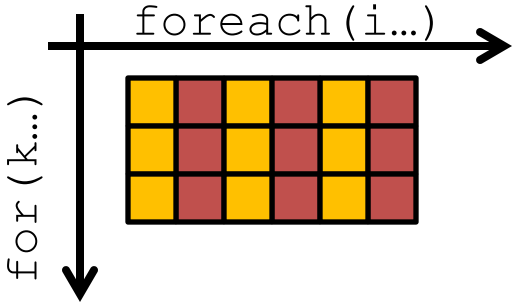
- Many different tiling choices

```
foreach( i∊[0,N) ){
   S(i);
}
```

```
foreach( i∊[0,N/T) ){
   foreach( k∊[0,T) ){
      S(k+i*T);
   }
}
```

Tiling with Dependencies

```
for(i=0; i<N; ++i){
  S(i);
}
```




```
foreach( i∈[0,N/T) ){
  for(k=0; k<T; ++k){
    S(k+i*T);
  }
}
```

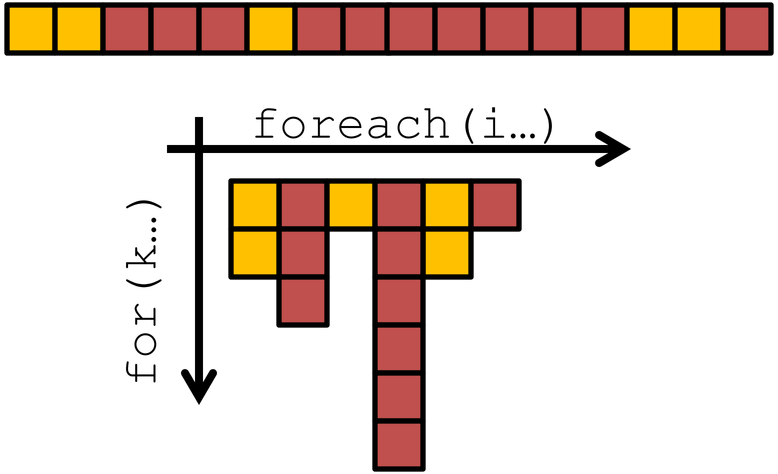
```
for(k=0; k<T; ++k){
  foreach( i∈[0,N/T) ){
    S(k+i*T);
  }
}
```

8

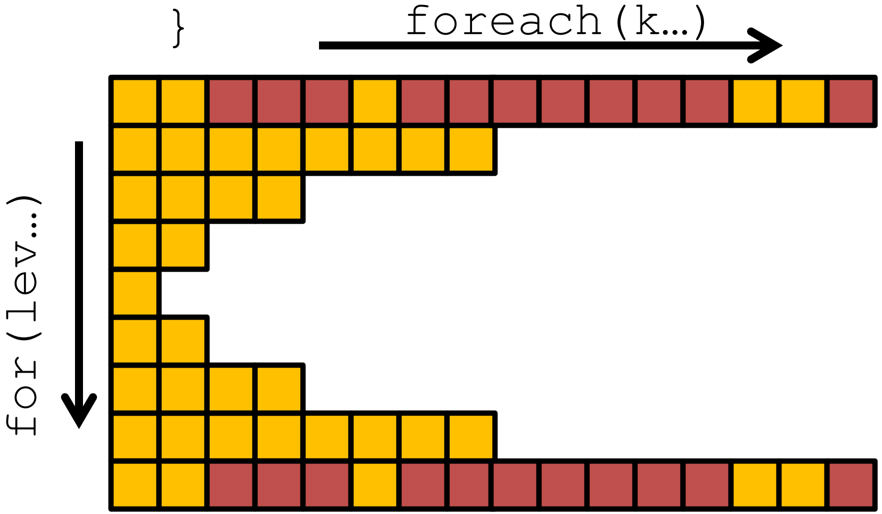
 9

Segments

```
foreach( i∊[0,M) ){
  for(k=0; k<L(i); ++k){
    S(i,k);
  }
}
```




```
for(lev…){
  foreach(k…){
    P(lev,k);
  }
}
```

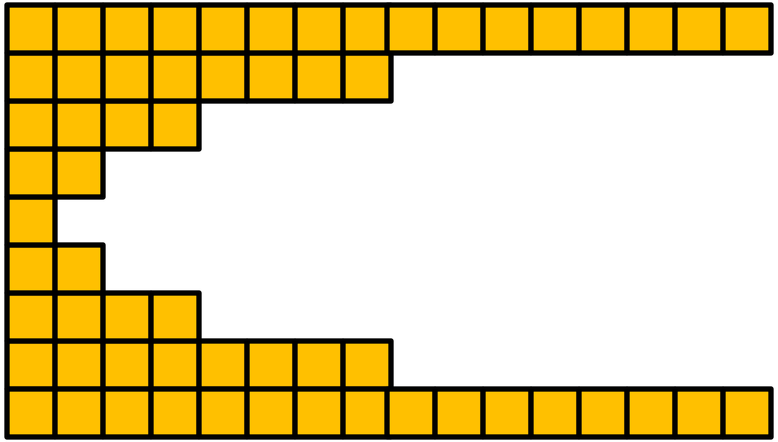


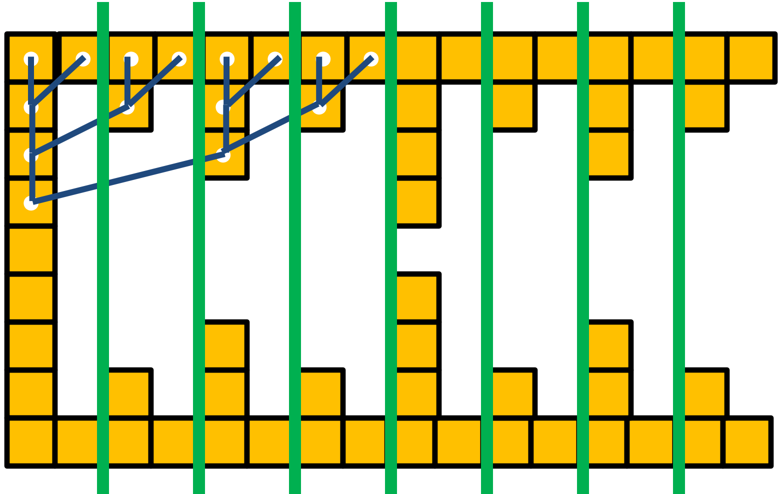
10

Hierarchies and Communication

```
for(lev…){
   foreach(k…){
      P(lev,k);
   }
}
```

- Reduction, scan, barrier
- Mipmaps, wavelets, multigrid
- Data distribution important



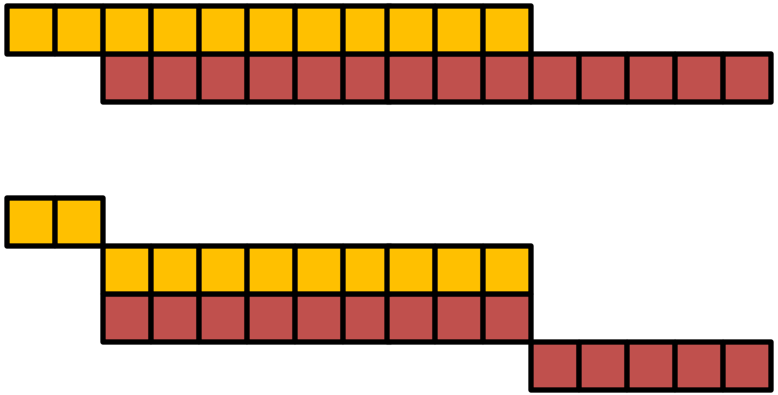


Loops, multiple accesses

12

Fusion of Two

```
foreach( i∈I_0 ){
   S_0(i);
}
foreach( i∈I_1 ){
   S_1(i);
}
```

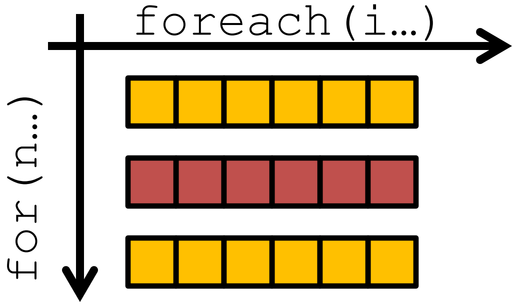


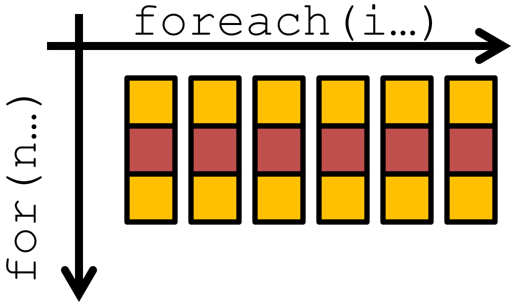
```
foreach( i∈I_0\I_1 ){
   S_0(i);
}
foreach( i∈I_0∩I_1){
   S_0(i); S_1(i);
}
foreach( i∈I_1\I_0 ){
   S_1(i);
}
```

Fusion of Many

```
for(n=0; n<N; ++n){
  foreach( i∈I_n ){
    S_n(i);
  }
}
```

```
…
foreach( i∈I_0∩…∩I_N-1 ){
  for(n=0; n<N; ++n){
    S_n(i);
  }
}
…
```





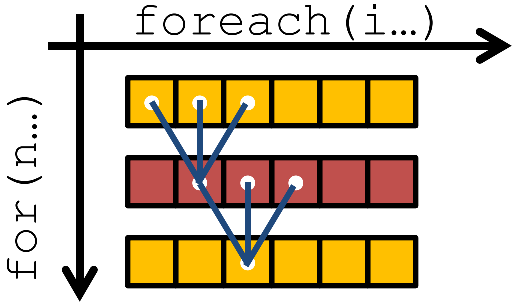
13

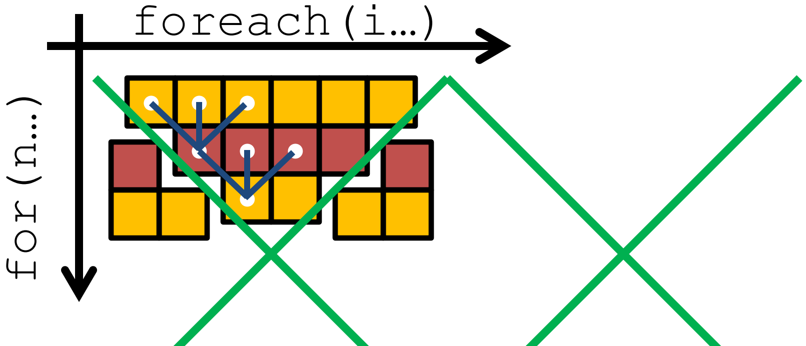
14

Fusion with Neighbor Dependencies

```
for(n=0; n<N; ++n){
  foreach( i∈I ){
    Sn(i);
  }
}
```

```
foreach(tile…){
  for(n…){
    foreach( i∈tile ){
      Sn(i);
    }
  }
}
```



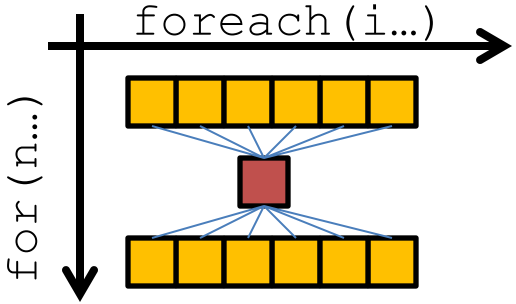


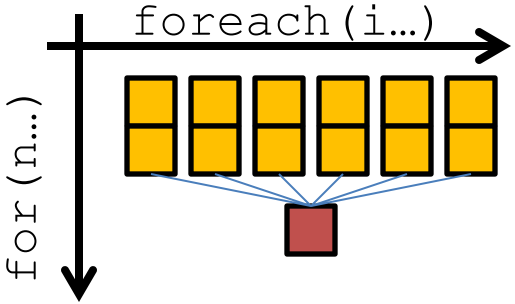
15

Fusion with Reductions

```
foreach( I→I ){…}
for( I→α ){…}
foreach( I→I ){… α …}
```

```
foreach( I→I ){…}
foreach( I→I ){…}
for( I→α ){…}
… α …
```





Branches

17

Extraction

```
foreach( i∈I ){
  if( cond0(i) ){
    S0(i);
  } else if( cond1(i) ){
    S1(i);
  } else {
    S2(i);
  }
}
```

```
foreach( i∈I0(cond0) ){
  S0(i);
}
foreach( i∈I1(cond1) ){
  S1(i);
}
foreach( i∈I2(cond2) ){
  S2(i);
}
```

Pre-Evaluation

18

```
foreach( i∈I ){
   if( cond0(i) ){
      S0(i);
   } else if( cond1(i) ){
      S1(i);
   } else {
      S2(i);
   }
}
```

```
compute(cond0[], cond1[]);

foreach( i∈I ){
   if( cond0[i] ){
      S0(i);
   } else if( cond1[i] ){
      S1(i);
   } else {
      S2(i);
   }
}
```

Predication

19

```
foreach( i∈I ){
  if( cond0(i) ){
    v= S0(i);
  } else if( cond1(i) ){
    v= S1(i);
  } else {
    v= S2(i);
  }
}
```

```
foreach( i∈I ){
  v0= S0(i);
  v1= S1(i);
  v2= S2(i);
  v= selector(i,v0,v1,v2);
}
```

Simple Rules

21

Golden Rules for High Performance Code

- Do not use branches
- Do not use loops
- Do not copy data

```
if(...) {}
for(...) {}
A= B;
```

22

Rules of Thumb for High Performance Code

- Avoid branches in loops — `if(…) {}`
- Avoid general loops — `for(…) {}`
- Avoid data copies — `A= B;`
- **Employ clever algorithms**

Conclusions

23

- Hardware(HW) – Software(SW)
  - Hardware designs highly specialized
  - Language constructs too general
  - Mismatch of SW to HW
- Algorithms
  - Important to avoid obvious donts
  - Some very efficient patterns exist
  - General cases execute poorly
- **Many things to be discovered !**

Application Specific Computing (ASC)

Robert Strzodka
Application Specific Computing
ZITI, Heidelberg University
http://asc.ziti.uni-heidelberg.de