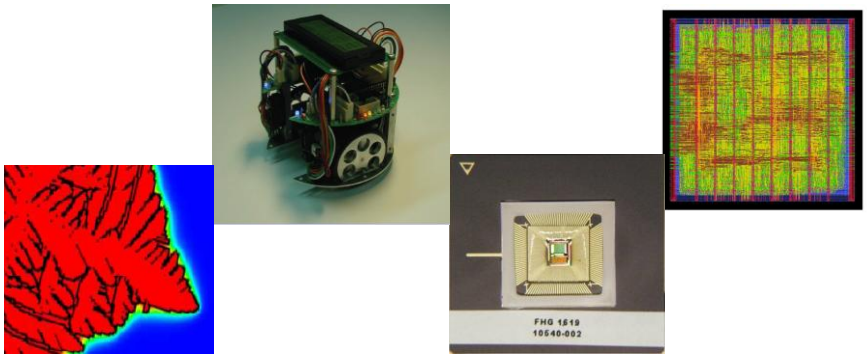
Transparenter Zugriff auf heterogene Architekturen für Stencil-Applikationen im Embedded und HPC-Umfeld



Dietmar Fey
Informatik 3 - Lehrstuhl für Rechnerarchitektur
Friedrich-Alexander-Universität Erlangen-Nürnberg

Übersicht

- Bedeutung Stencil-Applikationen
- Kurzer Überblick Stand der Technik
- Eigene Arbeiten
  - HPC-Umfeld: LibGeoDecomp
  - Embedded Umfeld: VHDL-Templates
- Zusammenfassung

Stencil Applikationen

- Stencil Applikationen:

  große Bedeutung für das wissenschaftliche Rechnen

  - Wärmeleitungsgleichung
  - Elektrodynamik, Elektromagnetismus
  - Strömungsmechanik
  - Lösung partieller Differentialgleichungen
  - Methode der Finiten Differenzen
  - Operatoren aus der Bildverarbeitung
  - ….

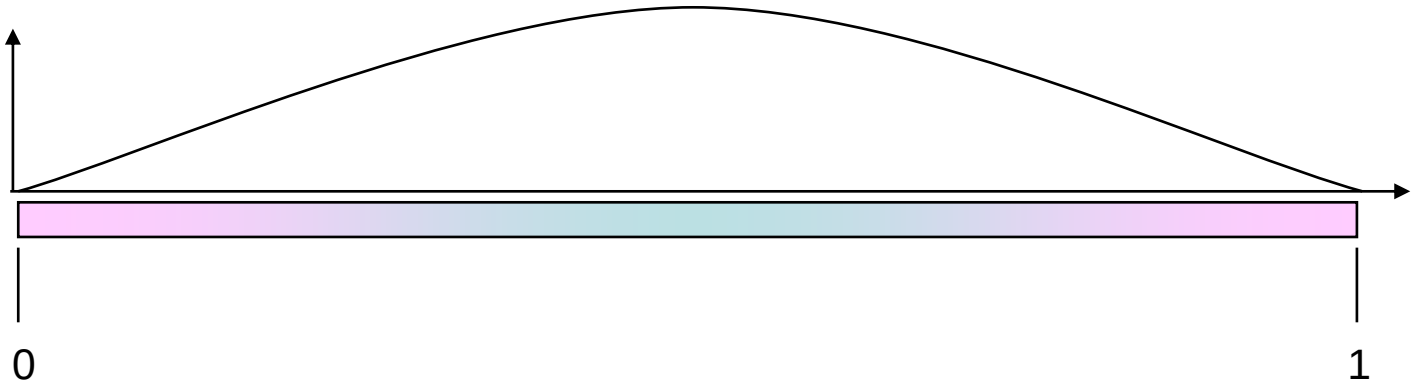
Stencil Applikationen

- Beispiel aus 1.Semester – Vorlesung Computational Engineering I im Bachelor-Studium CE
  - 1-dimensionale Wärmegleichung
    - Temperatur-Ausbreitung in einem dünnen Draht
  - beschreibbar durch Differential-Gleichung

$$\frac{\partial T}{\partial t} = k \cdot \frac{\partial^2 T}{\partial x^2}$$

Randbedingung:

- T(t,0) = 0, T(t,1) = 0 $\forall$ t
- T(0,x) = sin(πx) für $0 \leq x \leq 1$



Stencil Applikationen

- Überführung in Differenzen-Gleichung (s. Tafel)

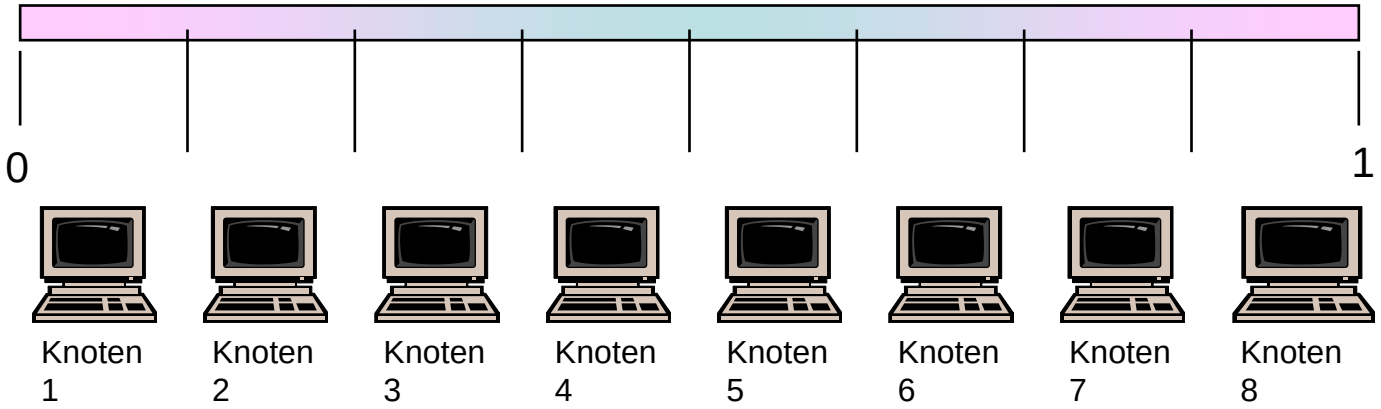
$$\frac{\partial T}{\partial t} = k \cdot \frac{\partial^2 T}{\partial x^2}$$

$$\frac{\partial T}{\partial t} = \frac{T_{t,i+1} - T_{t,i}}{\Delta t}; \frac{\partial T}{\partial x^2} = \frac{\left(T_{t,i+1} - T_{t,i}\right)/\Delta x}{\partial x} = \ldots$$

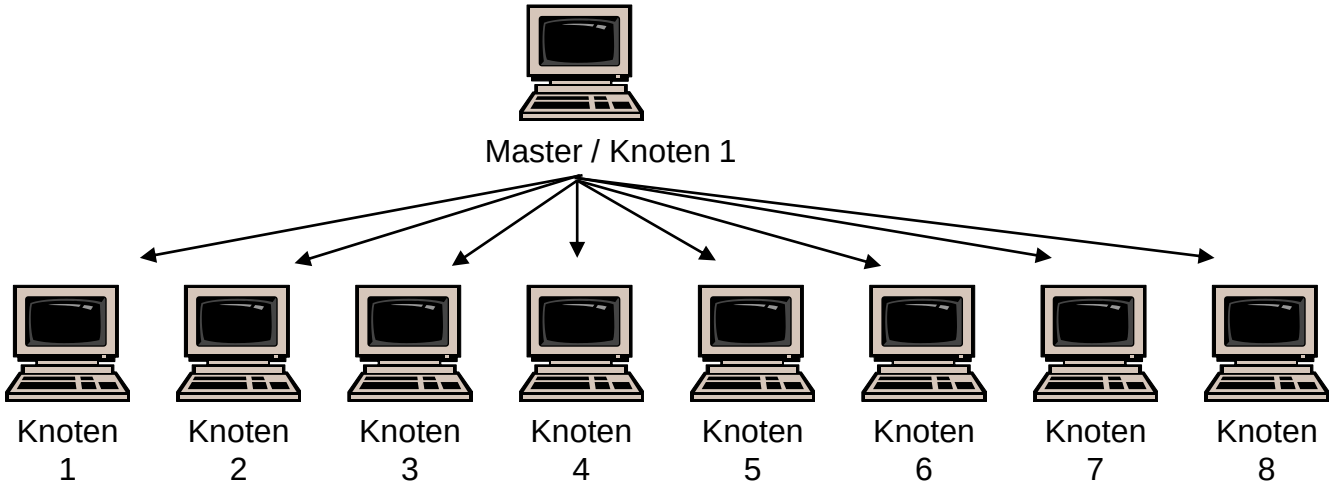
$$\Rightarrow \quad T_{t+1,i} = k \frac{\Delta t}{\Delta x^2}\left(T_{t,i+1} - 2T_{t,i} + T_{t,i-1}\right) + T_{t,i}$$

Stencil Applikationen

- Partitionierung für parallele Ausführung

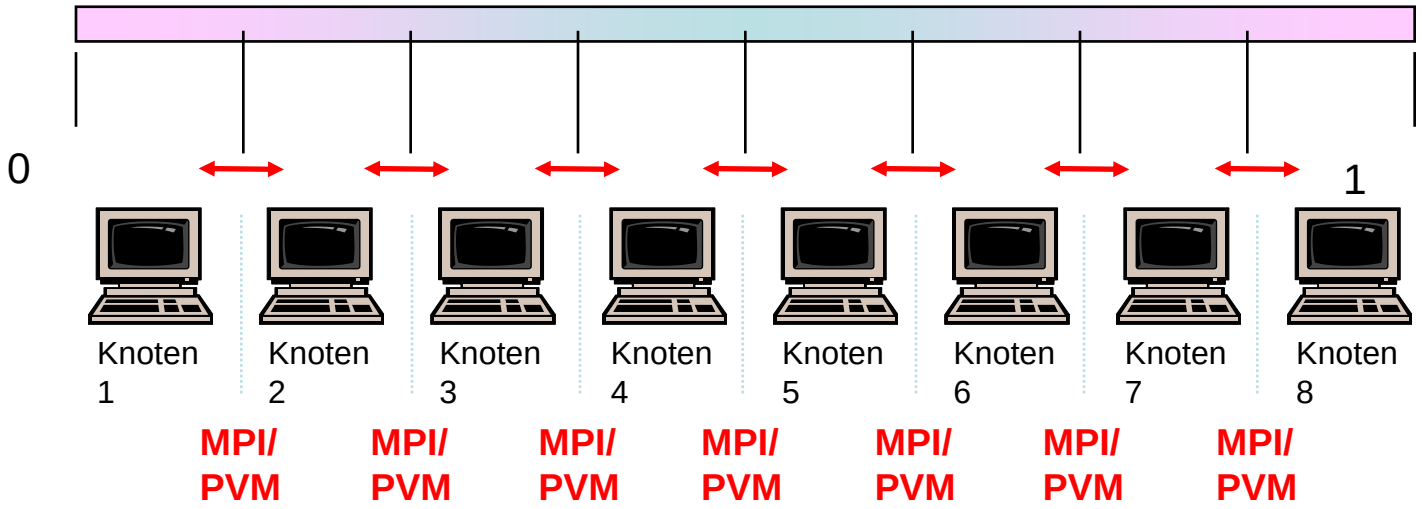


- Partitionierung durch Master



Stencil Applikationen

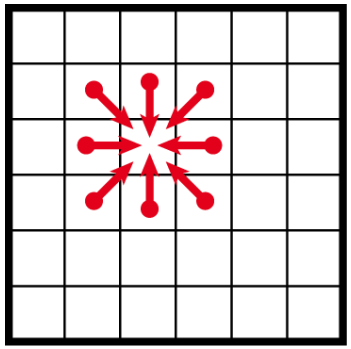
- Grenzen beachten und übergeben!

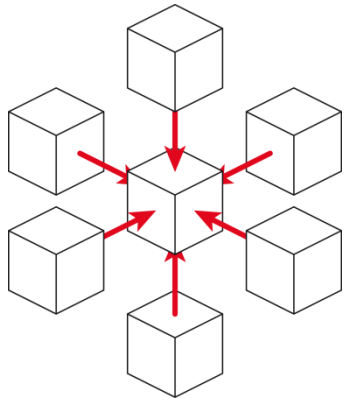


- Wie am Effizientesten die Berechnung in den Knoten im Hinblick auf die Kommunikation organisieren?

Stencil Applikationen

- Kennzeichen dieser Algorithmen
  - Daten-parallele Berechnung von 1D/2D/3D-Gitterpunkten
  - Nächste Nachbarschaftsberechnung
  - Zeit- und ortsdiskret





Stencil Applikationen

- Rechenintensiv
  - Beispiel 3D-Wärmegleichung

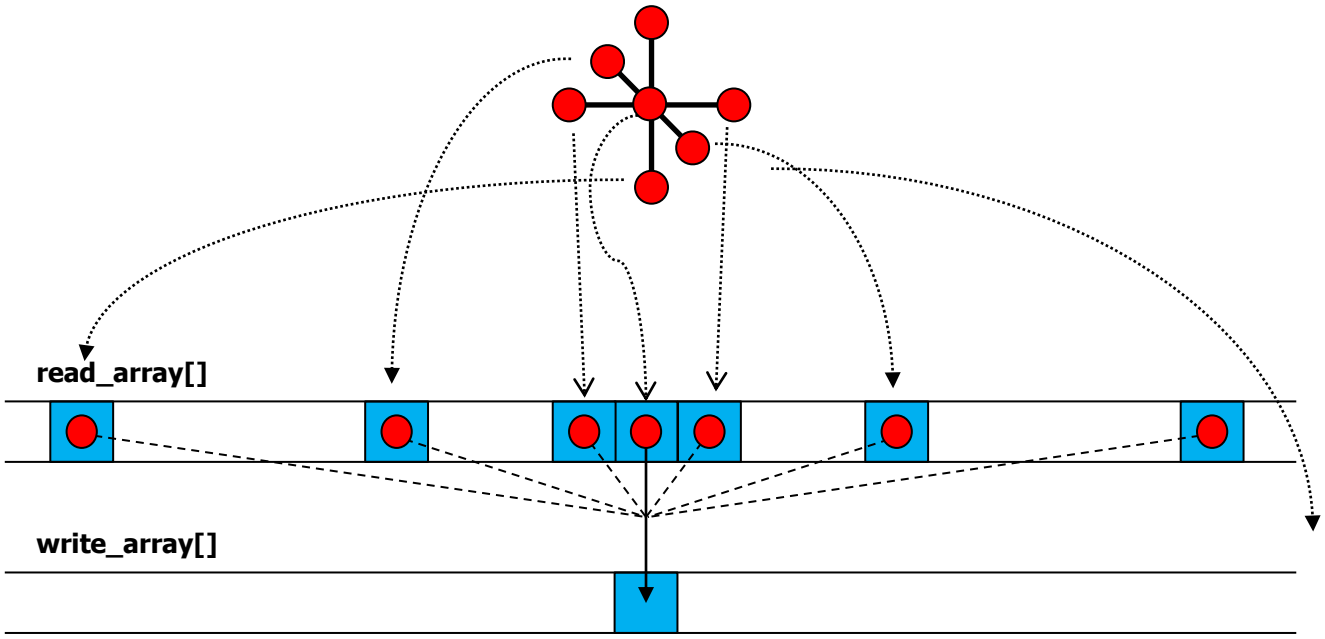
$$\frac{\partial A}{\partial t} = k \cdot \left(\frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2} + \frac{\partial^2 A}{\partial z^2} \right)$$

  - ergibt 7-Punkt Stencil
  - 4-fach verschachtelte Schleife über i,j,k und Zeit t

$$A^{t+1}(i,j,k) = C_o \cdot A^t(i,j,k) + C_1 \cdot \begin{pmatrix} A^t(i-1,j,k) + A^t(i+1,j,k) + A^t(i,j-1,k) \\ + A^t(i,j+1,k) + A^t(i,j,k-1) + A^t(i,j,k+1) \end{pmatrix}$$

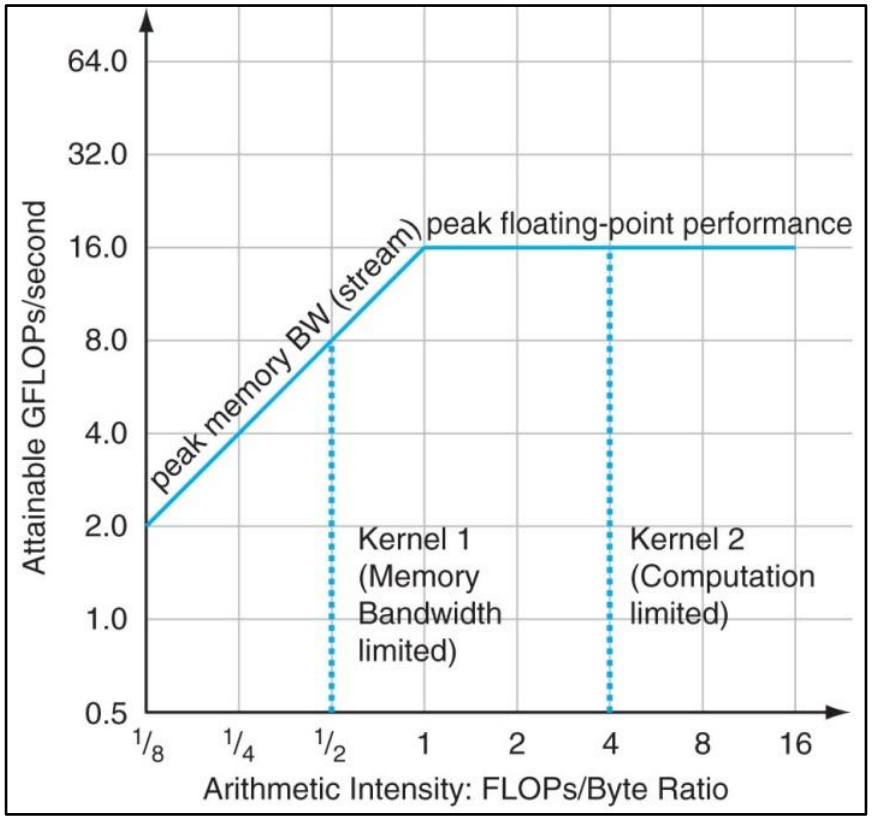
Stencil Applikationen

- Linearisierung des 3D-Datenvolumens im Speicher
  - 8 FP-Operationen
  - 24 Bytes bzw. 16 Bytes Speichertransfer



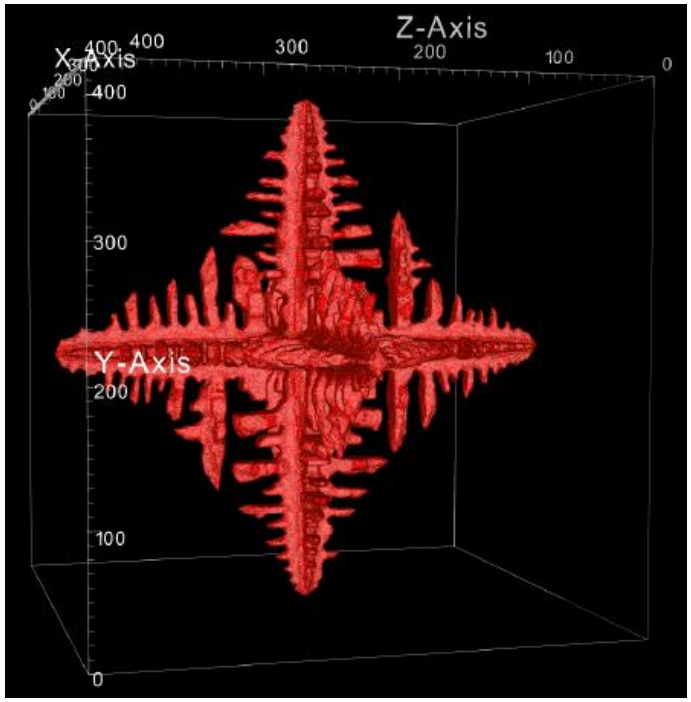
Stencil Applikationen

- Roofline-Modell
  - Operationelle / Arithmetische Intensität 0.33 bzw. 0.5
  - Stencil-Applikationen zumeist Speicherbandbreiten-begrenzt

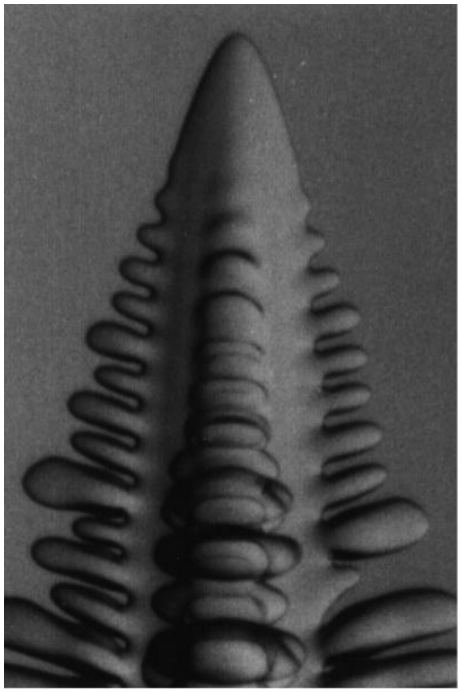


Stencil Applikationen

- Beispiel für Rechenintensität
  - Simulation Dendritenwachstum in abkühlenden Metallschmelzen / Kooperation FAU – FSU Jena, Materialwissenschaften



Simulation



Mikroskop

Stencil Applikationen

- Herausforderungen:
  - Rechenzeit ($1000^2 \rightarrow$ 8h, $2000^3 \rightarrow$ 7 Jahre?)
  - Berechnungen eng gekoppelt
  - Ideal geeignet für Parallelisierung der Stencil-Berechnung
  - Verschiedene parallele Algorithmen erforderlich für verschiedene parallele Architekturen
    - Z.B Multikern-CPUs oder GPUs

Stencil Applikationen

- Große Datenmengen in sehr kurzer Zeit berechnen
- Parallelisierungen notwendig mit Hardware-Beschleunigern
- HW-Beschleuniger
  - High Performance Computing (HPC)
    - GPUs, vereinzelt FPGAs
  - Embedded Anwendungen
    - FPGAs

  zusätzlich zu Multi-Core CPU
- → Heterogene Architektur

Stand der Technik

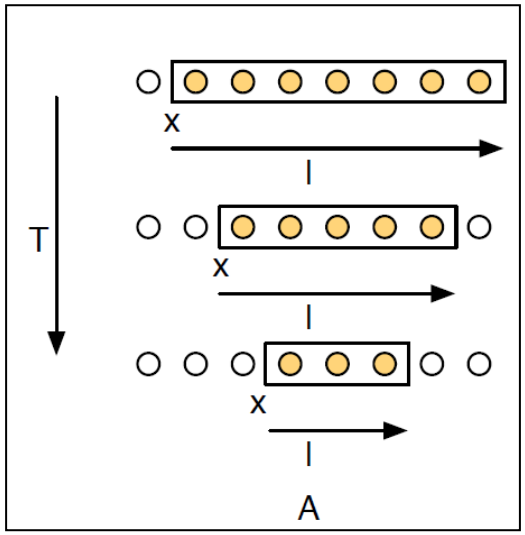
- Applikationsspezifische Kompilierung

  Justin Holewinski, Louis-Noel Pouchet and P. Sadayappan.
  High-Performance Code Generation for Stencil Computations on GPU Architectures
  ICS'12, June 25–29, 2012, San Servolo Island, Venice, Italy.

  Automatische Generierung von GPU-Kode für Stencil-Algorithmen

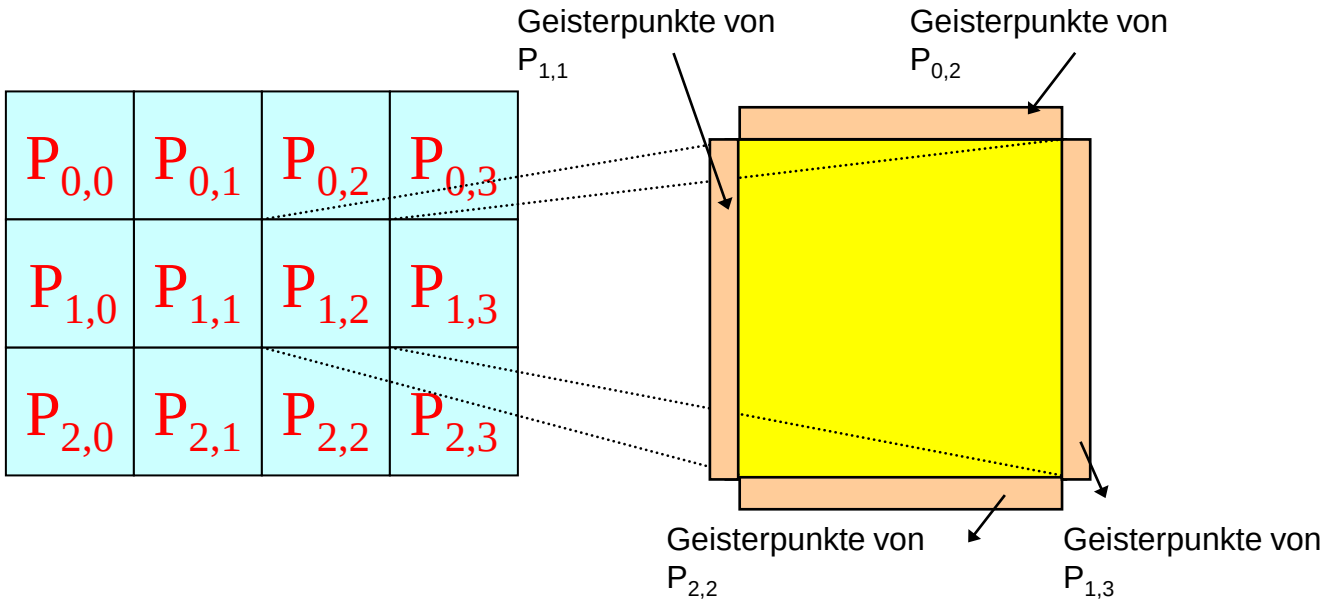
  - Stencil-Applikationen häufig Speicherbandbreiten-begrenzt
    - Gitterpunkte passen nicht mehr in Caches

- Lösung: „Time-tiling“
  - Schleifen-Verdrehung
  - Verlust an Nebenläufigkeit



Stand der Technik

- Lösung für Verlust an Nebenläufigkeit:
  - Überlappende Bereiche
    - Geisterzonen, Geisterzonen-Optimierung
  - Preis
    - Redundante Berechnungen vs. Einsparen von Kommunikation bzw. Cachetransfers



Stand der Technik

- Analyse „Latency hiding“
  - Das Vermehren der Geisterpunkte hat drei Effekte:
    - wachsende Nachrichtenlängen / Speicherstellen
    - verminderte Häufigkeit von Nachrichten / Cachetransfers und
    - zusätzliche redundante Berechnungen
- Paralleler Overhead
  - als Funktion der Anzahl der Geisterpunkte bestimmen
  - Wenn es *k* Geisterpunkte auf jeder Seite gibt,
    - so werden alle *k* Iterationen Nachrichten der Länge *k* * zu und von den Nachbarn gesendet
      - * (entspricht der k-fachen Speichermenge für ein auszutauschendes Datum)

Stand der Technik

- Sei λ die Latenz und β die Bandbreite
  - Dann dauert das Senden einer Nachricht der Länge $\lambda + k / \beta$ Zeiteinheiten

- Die Anzahl zusätzlicher redundanter Berechnungen ist

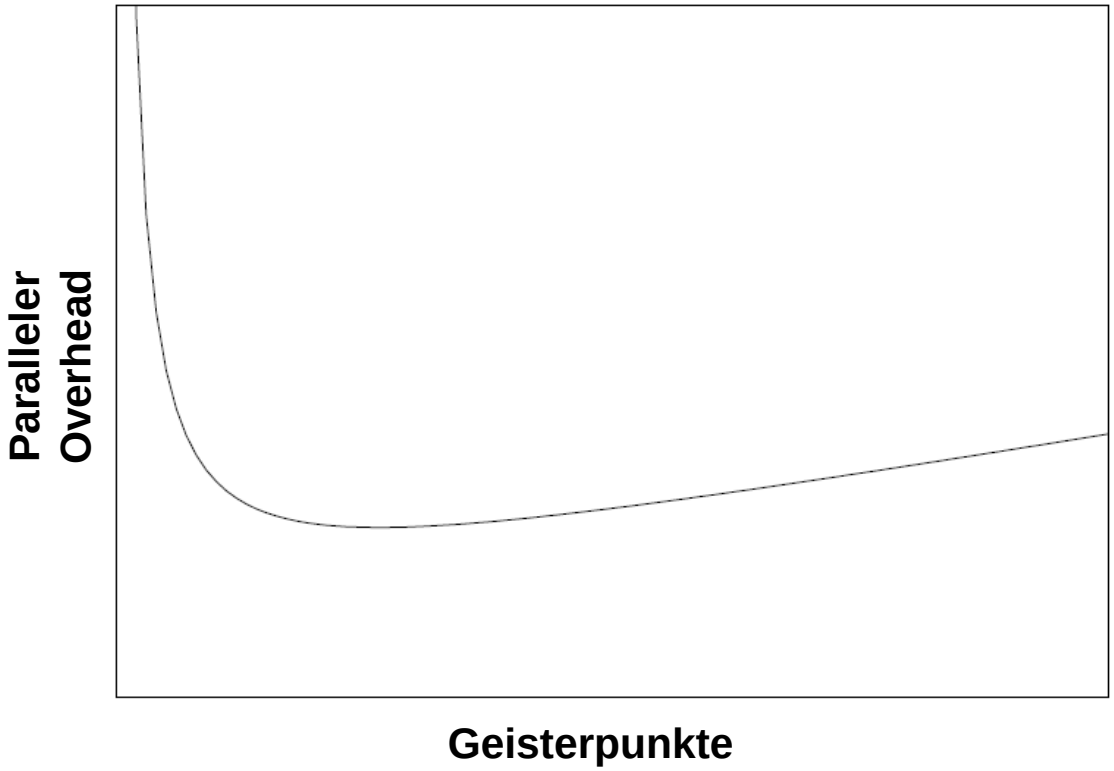
$$\sum_{i=1}^{k-1} i = k(k-1)/2.$$

- Sei χ die Dauer einer solchen Berechnung, dann beträgt der parallele Overhead pro Prozess und pro Iteration

$$\frac{2(\lambda + k/\beta) + \chi k(k-1)/2}{k} = \frac{2\lambda}{k} + \frac{2}{\beta} + \frac{\chi(k-1)}{2}$$

Stand der Technik

- Verlauf Overhead vs. Anzahl Geisterpunkte *k*



Stand der Technik

- Compiler-Algorithmen für überlappende Geisterzonen-Optimierung (Holewinski et.al.)
  - Stencil DSL (Domain Specific Language)
    - Grids
    - Fields
    - Stencil Functions

```
1  real A[M];
2  real A_tmp[M];
3
4  copy(A_tmp /* dest */, A /* src */);
5
6  for (n = 0; n < T; ++n) {
7    A[0] = A_tmp[0];
8    for (i = 1; i < M-1; ++i) {
9      A[i] = 0.333 * (A_tmp[i-1] + A_tmp[i] + A_tmp[i+1]);
10   }
11   A[M-1] = A_tmp[M-1];
12   swap(A_tmp, A);
13 }
```

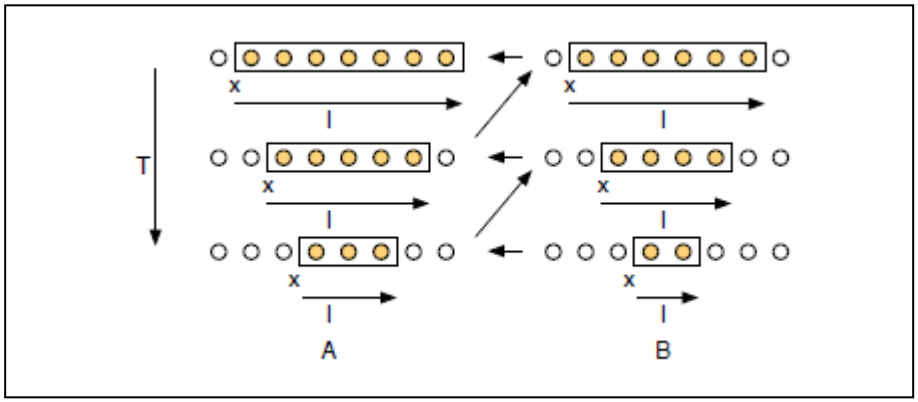
$$
\begin{aligned}
G &= [0, M-1], 64 \\
A_G &: \text{real} \\
f_1(A_G)[0] &= A[0] \\
f_2(A_G)[1..M-2] &= 0.333 * (A[-1] + A[0] + A[1]) \\
f_3(A_G)[M-1] &= A[0]
\end{aligned}
$$

Stand der Technik

- Benutzer macht Vorgaben neben Grid, Field, Stencils
  - Überlappungsbreite: „Time-tile size“
  - Thread Mächtigkeit: #Zellen pro Thread
  - Thread-Block-Größe: #Threads / Block

Daraus wird automatisch OpenCL-Kode erzeugt

- Beispiel für automatische Bestimmung Überlappungsbreite



$$f_A(B_G)[1..N-2] = B[-1] + B[0]$$
$$f_B(A_G)[1..N-2] = A[0] + A[1]$$

$$(\vec{x}_A, \vec{l}_A) = (x_0 - 2, l_0 + 5)$$
$$(\vec{x}_B, \vec{l}_B) = (x_0 - 2, l_0 + 4)$$

Stand der Technik

- Optimierung über Auto-Tuning

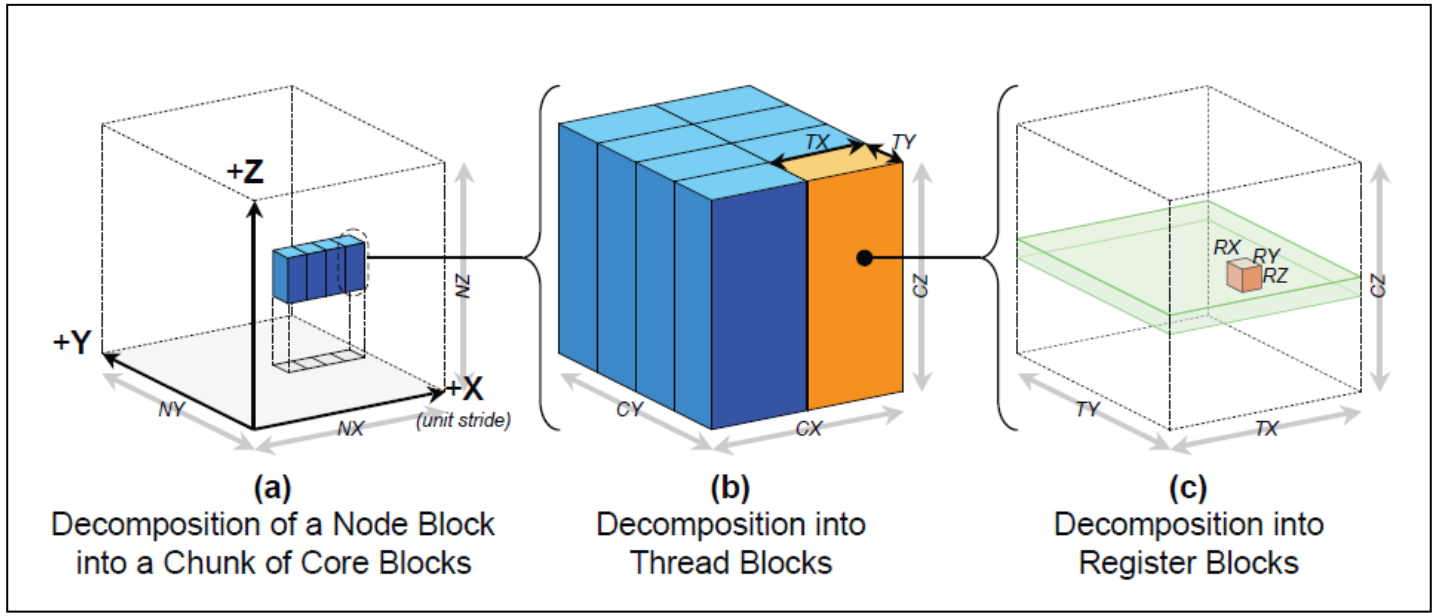
  Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures

  Kaushik Datta et.al., Lawrence Berkeley National Laboratory, UC Berkeley

  - NUMA Affinität
  - Array Padding
  - Core/Register Blocken
  - Prefetching
  - SIMD-Ausnutzung
  - Spezielle algorithmische Transformationen für Stencils
    - Thread Blocking
    - Ring-Warteschlangen (circular queues)

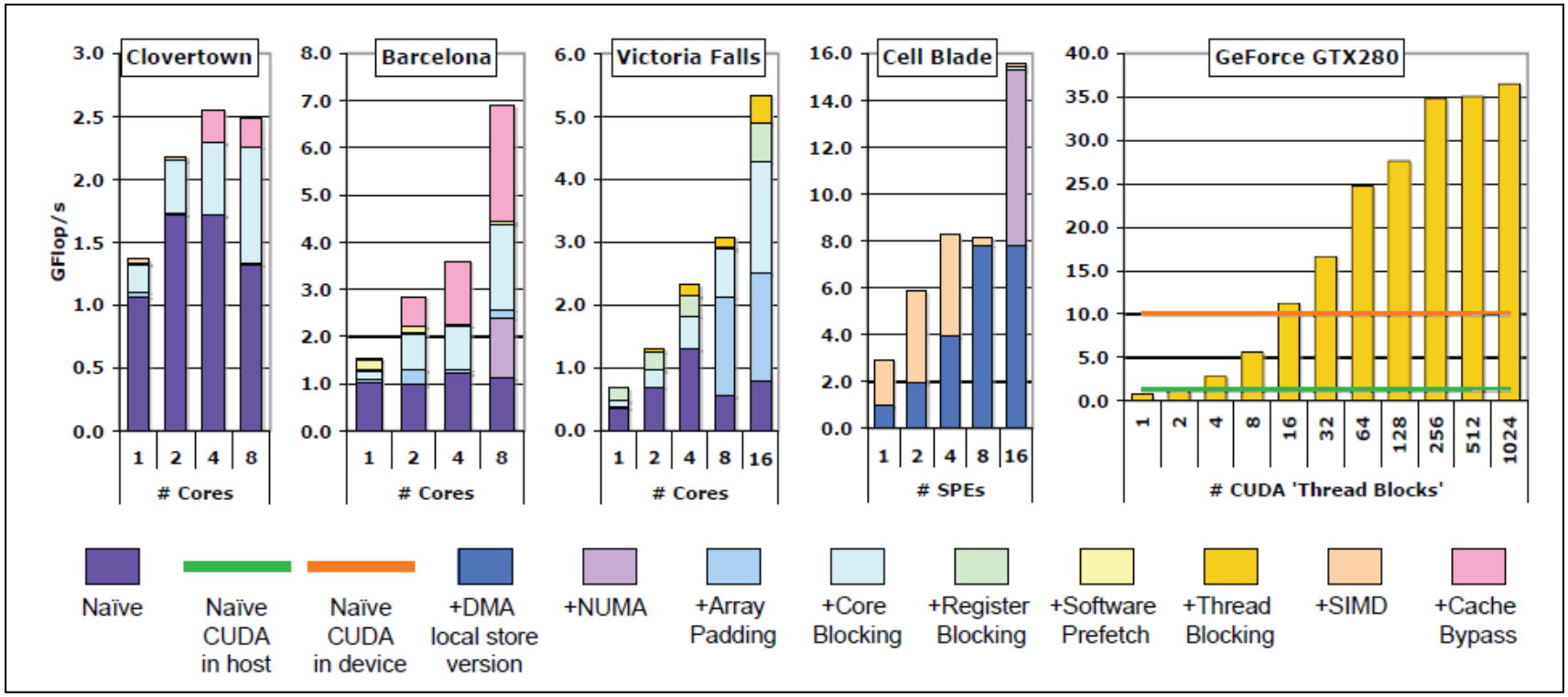
Stand der Technik

- Hierarchische Blockzerlegungen
  - für Knoten, Kerne, Threads, Register



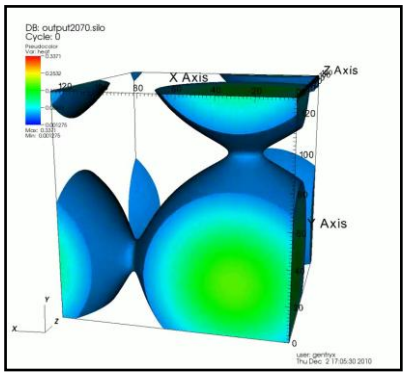
Stand der Technik

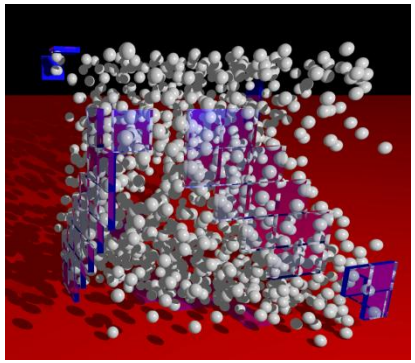
- Optimalen Parametersatz durch Testläufe und Messungen bestimmen
  - Für verschiedene Architekturen durchgeführt

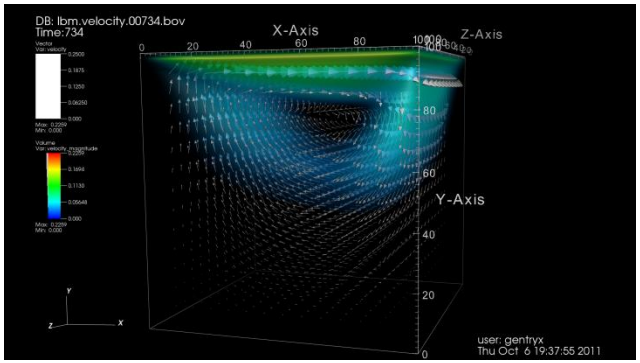


LibGeoDecomp – Stencil-Bibliothek für HPC-Umfeld

- Bibliothek für Kodes Prinzip Geometrische Zerlegung
- Auto-Parallelisierung für Simulationen
- Iterative Algorithmen mit lokalen Wechselwirkungen, z.B.
  - Stencil Kodes
  - Kodes für Vielteilchenmodelle
  - Particle-in-Cell Kodes
- Anwendbar für Mehrkern-Prozessoren, GPUs, Cluster, …
- Beispiele: Jacobi Iteration, granulares Gas, LBM





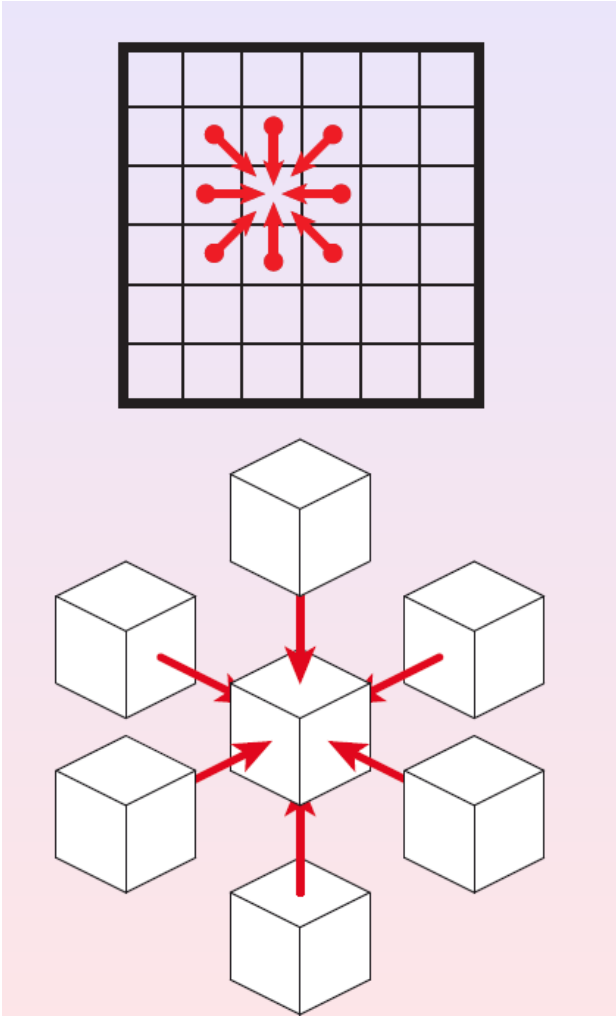


LibGeoDecomp: Grundlegende Funktionsweise

- Ziel: wiederverwendbare, effiziente Lösungen
- Einfach anwendbar
  - **Automatisches Weiterleiten der Zugriffe auf Nachbarzellen**
  - **Ersetzt Schleife über diskreten Raum/Zeit** durch Bibliotheksaufruf
- Implementiert als C++ Template Library
- Anwender stellt ein Modell bereit
- Bibliothek übernimmt Parallelisierung
  - Plugins für verschiedene Architekturen
  - Topologie-Erkennung wählt passende Plugins für Kommunikation
  - Parallelisierung transparent für Benutzer

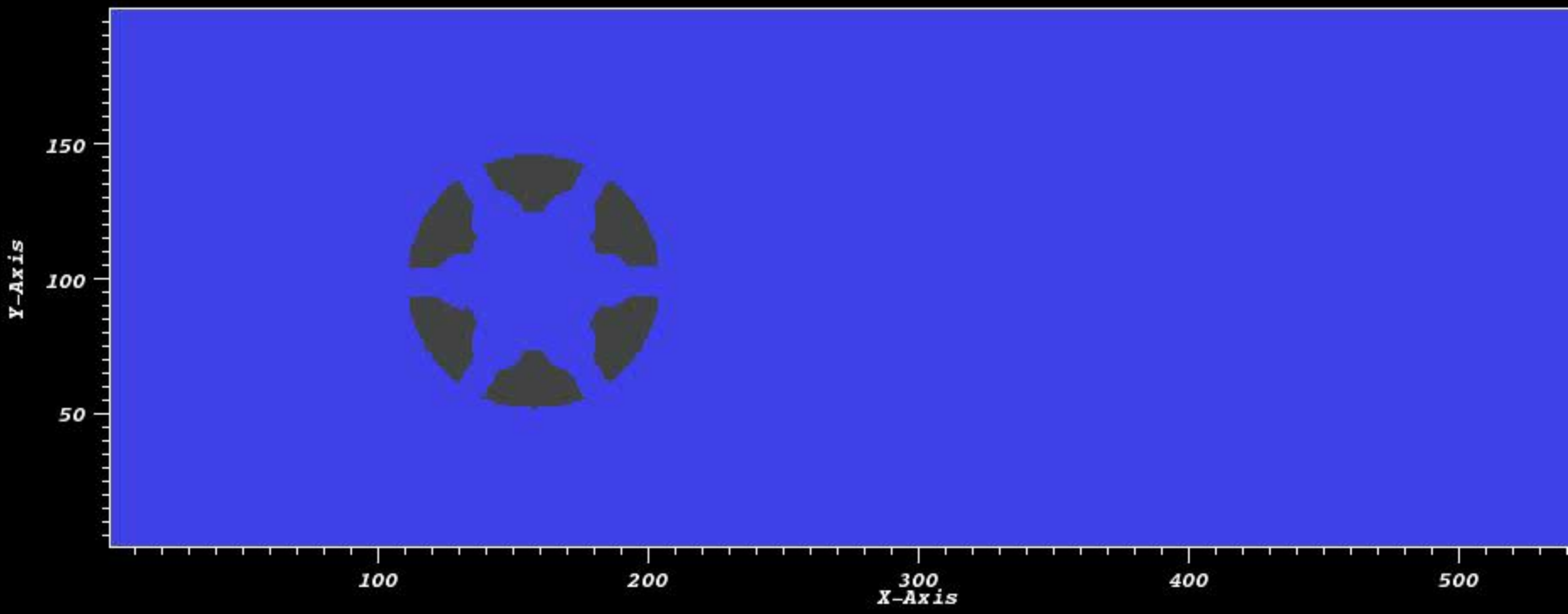
LibGeoDecomp: How does it work?

- Lattice Boltzmann Method Toolkit
  - 2D/3D Fest/Flüssig Interaktion
  - Verschiedene Randbedingungen:
    - Free-slip, No-slip, Moving Wall
    - Zou He (Druck, Geschwindigkeit)
  - Verschiedene LBM-Kernels definiert
    - Implementierung anhand analytisch bekannter Lösungen verifiziert
  - Ein-/Ausgabeunterstützung durch Bibliothek



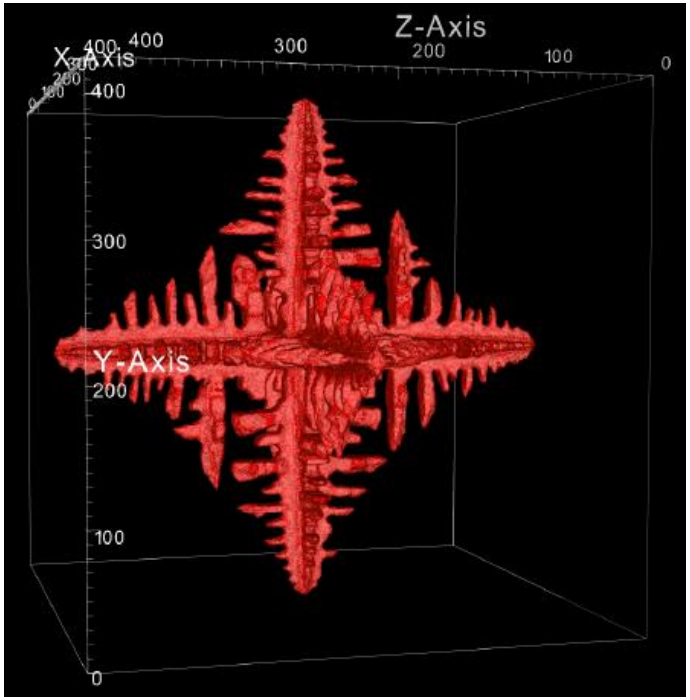
.00000.bov

tude

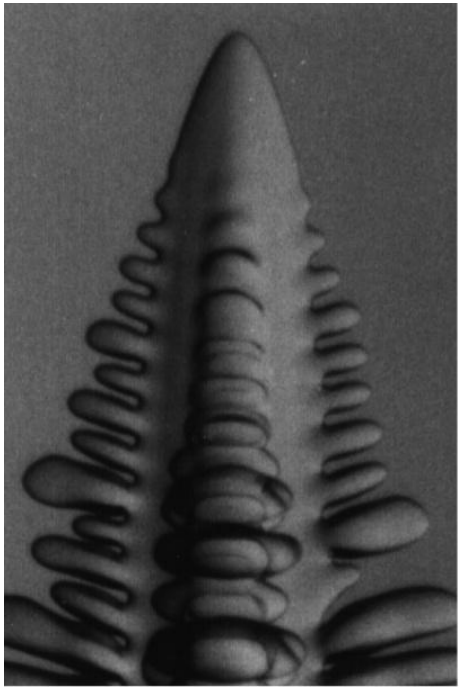


Stencil Applikationen

- DendSim3



Simulation



Mikroskop

LibGeoDecomp – DendSim3

- Möglichkeiten der Parallelisierung
  - Manuell
  - Neuimplementierung mit PSE (z.B. Cactus)
  - Neuimplementierung mit paralleler Programmiersprache
  - Portieren mit LibGeoDecomp

- Portieren mit LibGeoDecomp:
  - Gesamte Kodebasis: 5000 LOC (Lines Of Code)
  - Portierung: 100 LOC zu ändern
  - Video:
    - 2 TB Output
    - 7 Tage für Rechnen, 3 Tage Visualisierung
  - Aktiv im Einsatz seit 2011

300

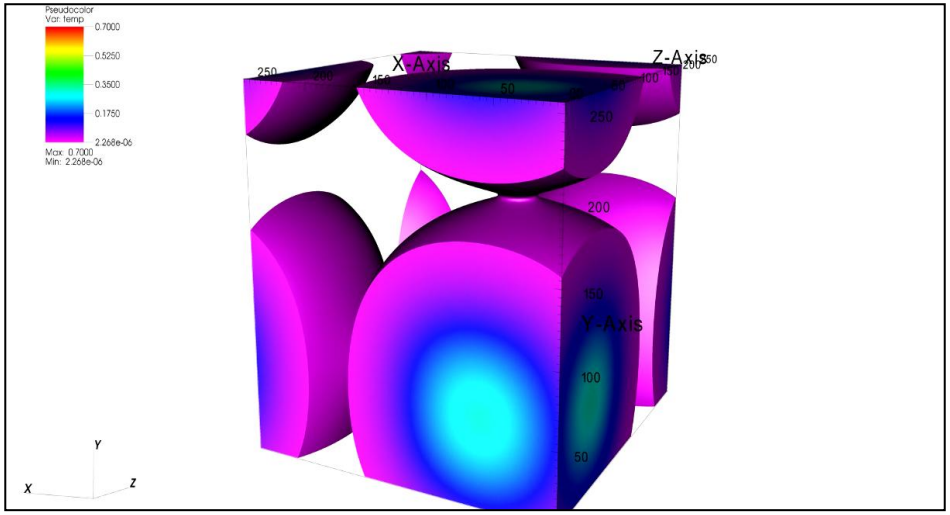
Y-Axis

200

100

3D Jacobi: NaïveCode

- Wäremeleitung
  - Periodische Randbedingungen
  - Bekanntes, weit-verbreitetes Beispiel
- Erscheint einfach
  - Viel Potential für Optimierung
  - Viele Fallstricke



3D Jacobi: NaïveCode (cont.)

```
#define GET(X, Y, Z) ( * gr idOld ) [ \
      ( ( z + Z + gridDim ) % gridDim ) * gridDim * gridDim + \
      ( ( y + Y + gridDim ) % gridDim ) * gridDim + \
      ( ( x + X + gridDim ) % gridDim ) ]
#define SET( ) ( * gridNew ) [ \
      ( ( z + gridDim ) % gridDim ) * gridDim * gridDim + \
      ( ( y + gridDim ) % gridDim ) * gridDim + \
      ( ( x + gridDim ) % gridDim ) ]

for ( int z = 0; z < gridDim ; ++z ) {
      for ( int y = 0; y < gridDim ; ++y ) {
            for ( int x = 0; x < gridDim ; ++x ) {
               SET( ) =
                  (GET( 0 , 0 , -1) + GET( 0 , -1, 0) + GET(-1, 0 , 0) +
                  GET( 1 , 0 , 0) + GET( 0 , 1 , 0) + GET( 0 , 0 , 1 ) ) * (1.0 / 6 . 0 ) ;
            }
      }
}
```

- 0.10 GLUPS, 1 Thread
- Intel Core i7-3610QM @2.30GHz (Ivy Bridge)
- Matrixgröße: 5123 cells

3D Jacobi: LibGeoDecomp Modell

```
#pragma omp parallel for
for ( int z = 0; z < gridDim ; ++z ) {
    for ( int y = 0; y < gridDim ; ++y ) {
        for ( int x = 0; x < gridDim ; ++x ) {
            SET( ) =
                (GET( 0 , 0 , -1) + GET( 0 , -1, 0) + GET(-1, 0 , 0) +
                 GET( 1 , 0 , 0) + GET( 0 , 1 , 0) + GET( 0 , 0 , 1 ) ) * (1.0 / 6 . 0 ) ;
        }
    }
}
```

- 0.37 GLUPS, 4 Threads
- Intel Core i7-3610QM @2.30GHz (Ivy Bridge)
- Speed-up 3.6

3D Jacobi: LibGeoDecomp Modell

```
class Cell {
public :
    typedef Stencils : : VonNeumann<3 , 1> Stencil ;
    typedef Topologies : : Torus <3 >:: Topology Topology ;
    class API : public Cell APITraits : : Fixed { } ;

Static inline unsigned nanoSteps ( ) { return 1; }

Inline explicit Cell ( const double& v=0) : temp ( v ) { }

#define GET(X, Y, Z) hood [ FixedCoord <X, Y, Z > ( ) ] . temp
#define SET( ) temp
    template<typename COORD_MAP>
    void update ( const COORD_MAP& hood , const unsigned& nanoStep )
    {
        SET( ) = (GET( 0 , 0 -1) + GET( 0, -1, 0) + GET(-1, 0 , 0) + GET( 1 , 0 , 0) + GET( 0 , 1 , 0)  +
                  GET( 0 , 0 , 1 ) ) * (1.0 / 6 . 0 ) ;
    }
    double temp ;
} ;
```

3D Jacobi: LibGeoDecomp Simple

```
#include <libgeodecomp .h>

class Cell{ . . . } ;
class CellInitializer{ . . . } ;

int main ( int argc , char * argv [ ] )  {
      CellInitializer * init = new CellInitailzier( gridDim , steps ) ;
      SerialSimulator <Cell> sim( init ) ;
      sim.run ( ) ;
      return 0;
}
```

- 0.40 GLUPS, 1 Thread
- 0.50 GLUPS, 4 Threads
- Intel Core i7-3610QM @2.30GHz (Ivy Bridge)

3D Jacobi: LibGeoDecomp Cache Blocking

```
#include <libgeodecomp . h>

class Cell { . . . } ;
class Cell Initializer { . . . } ;

int main ( int argc , char * argv [ ] )  {
    CellInitializer * init = new CellInitiaizier( gridDim , steps ) ;
    CacheBlockingSimulator<Cell> sim(init, 3, Coord<2>(270, 30));
    sim.run ( ) ;
    return 0;
}
```

- 1.04 GLUPS, 4 Threads
- Intel Core i7-3610QM @2.30GHz (Ivy Bridge)

3D Jacobi: LibGeoDecomp Cuda Schnittstelle

```
#include <libgeodecomp . h>

class Cell { . . . } ;
class CellInitializer { . . . } ;

int main ( int argc , char * argv [ ] )  {
    CellInitializer * init = new CellInitializier( gridDim , steps ) ;
    CudaSimulator<Cell> sim(init);
    return 0;
}
```

- 2.48 GLUPS, Tesla C2050
- 4.87 GLUPS, Tesla K20
- 47x schneller als der naïve Kode

Drei „Swimlanes“ zu Exascale

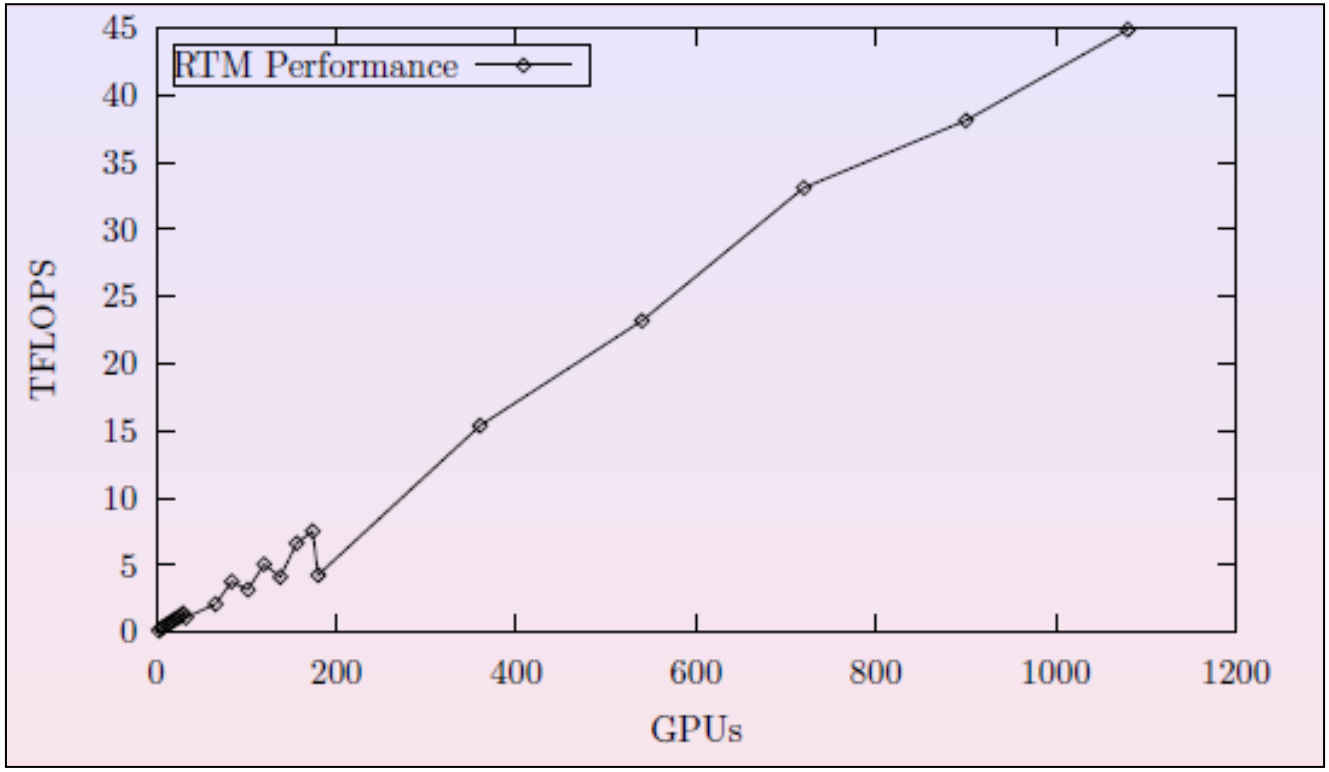
- **Multi-cores (e.g. Intel Xeon; SuperMUC am LRZ)**
- **GPUs (e.g. Nvidia K20x; Titan am ORNL)**
- **Embedded CPUs (z.B. PowerPC; Juqueen am FZ-Jülich)**

- Kategorien nach Horst Simon, stellvertretender Direktor am Lawrence Berkeley National Laboratory (LBNL)

- LibGeoDecomp unterstützt alle drei Architekturrichtungen

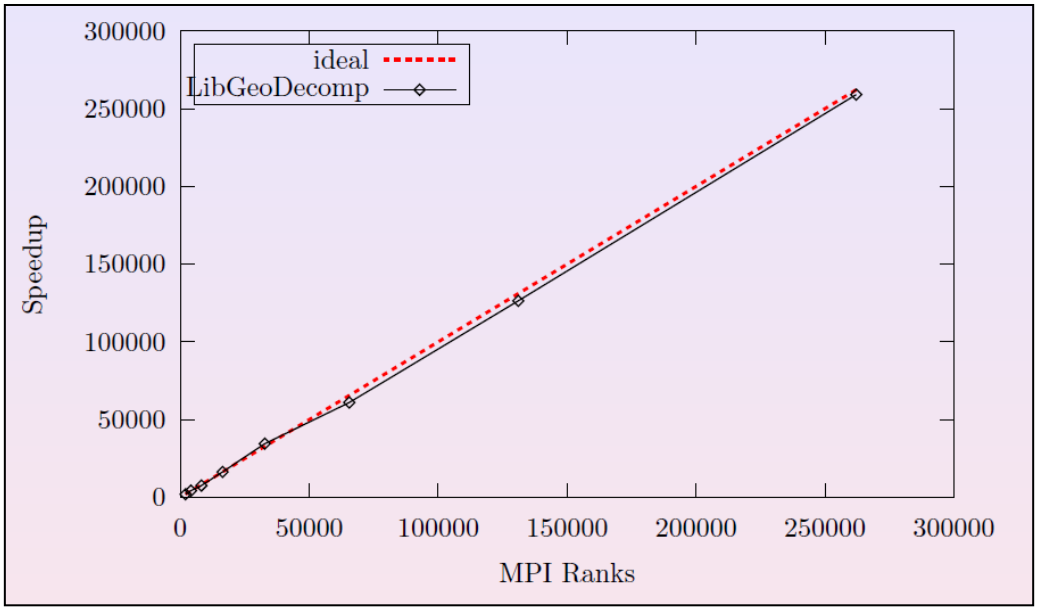
Schwache Skalierung Reverse Time Migration

- Tsubame (bis zu 1080 GPUs, Tesla M2050)
  - Gittergröße = (1070; 256; 512) × GPUs1/3
  - Effizienz gemäß „Weak“-Skalierung 90%



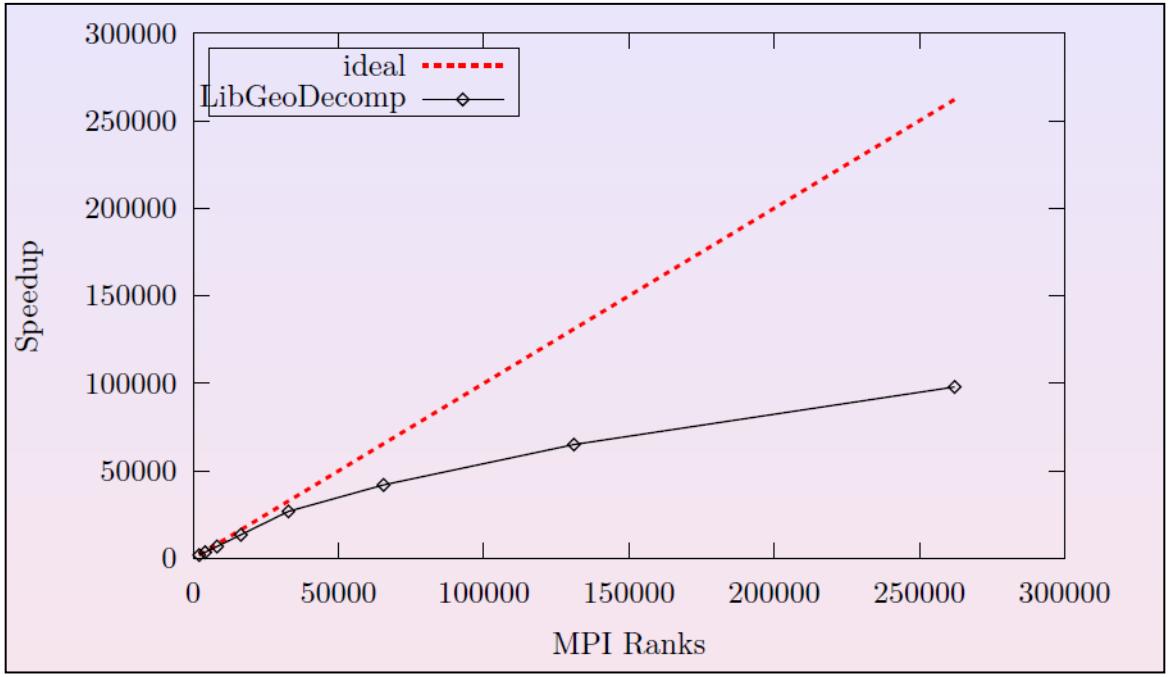
Vielteilchenkode auf Juqueen

- Supercomputing mit PowerPC-Prozessoren
  - Schwache Skalierung
    - Skaliert bis zu 262k MPI Ranks
    - Bis zu 112.893 GigaPartikel simuliert
  - Beide Tests mit 64 Ranks per Konten ausgeführt
    - 4x Überladung um 4x SMT der Kerne auszunutzen (wg. In-order)



Vielteilchenkode auf Juqueen

- Starke Skalierung bis zu 262k MPI Ranks
  - Fixiert bei 90M Partikel
  - Sehr gute Skalierbarkeit:
    - Last verteilt von 1 Rank bis zu 262k Ranks
    - Immer noch 37% des Optimums



Zusammenfassung LibGeoDecomp

- Frei verfügbare Open Source Software
- unterstützt
  - Stencil codes
  - Vielteilchensimulationen
  - „Particle-in-Cell“ Kodes
- Skaliert von Smart Phones bis zu Superrechnern
- Getestet mit bis zu
  - 488k Kernen
  - 262k MPI Knoten
  - 65 TB RAM
  - 112.893 Giga Partikeln

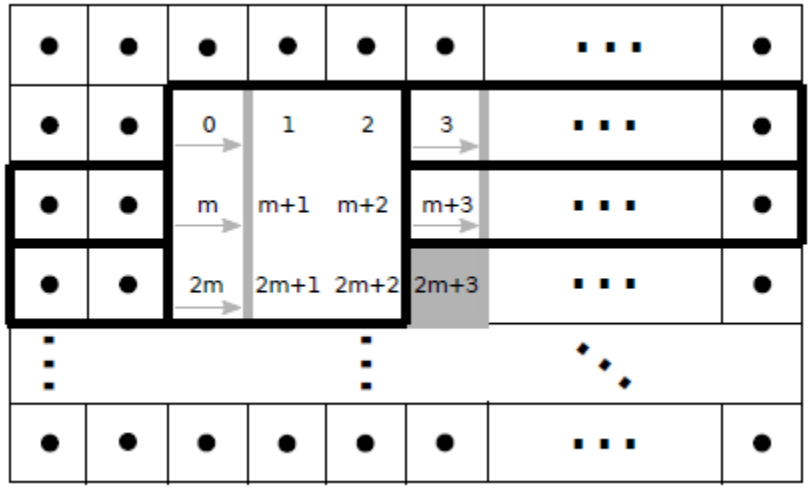
Verarbeitung von Stencils im Embedded-Umfeld

- Smart Camera
  - Integration von verarbeitender Logik direkt in Kamera
  - Kosten sparen (kein PC)
  - Datenreduzierung in der Kamera

- Hohe Anforderungen an bildverarbeitende Algorithmen
  - Realzeit-Anforderung
  - Parallelverarbeitung mit eigenen Parallelrechnerarchitekturen

Verarbeitung von Stencils im Embedded-Umfeld

- Optimierung der Datenpufferung
  - Daten Streaming für 2D Stencil Codes (Full Buffering (FB))



→ Sliding Window Verarbeitungsmethode

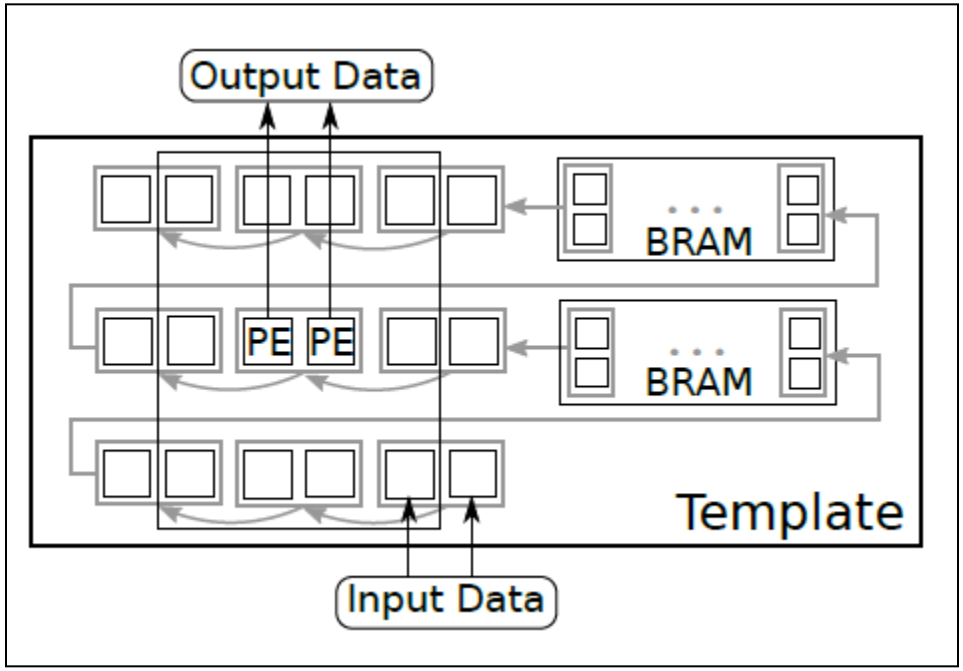
→ keine redundante Zellzugriffe

→ zusätzliche Speicher erforderlich

Verarbeitung von Stencils im Embedded-Umfeld

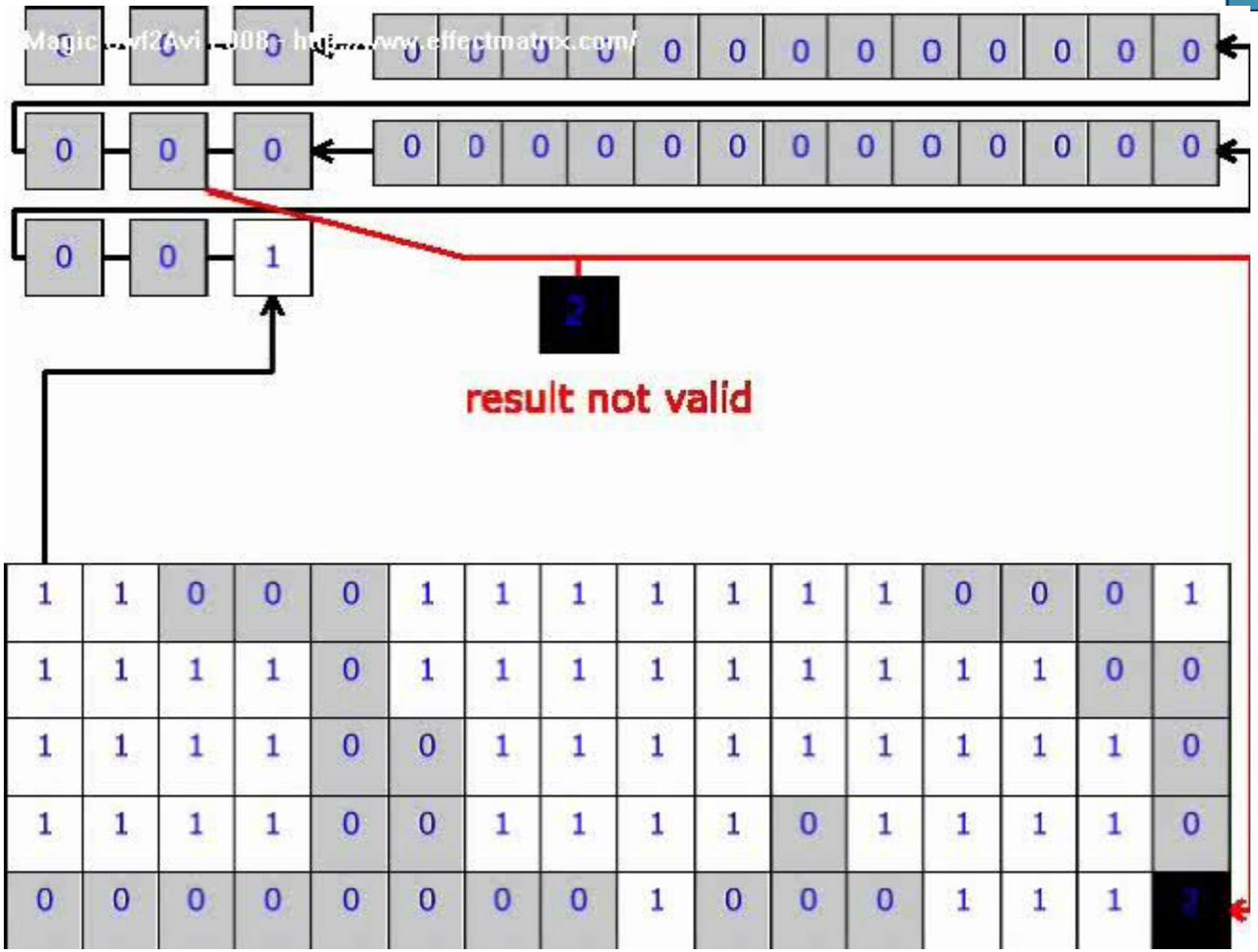
- FB mit adaptierbarer Parallelisierung

- Parallelisierung hängt von der Bandbreiten-Kapazität der E/A-Schnittstelle des FPGA ab




```
for t in TIME
    for s in SPACE/2
      parallel_exec
        // space unrolling
        update_cell(2*s+0,t)
        update_cell(2*s+1,t)
      end
    end
end
```

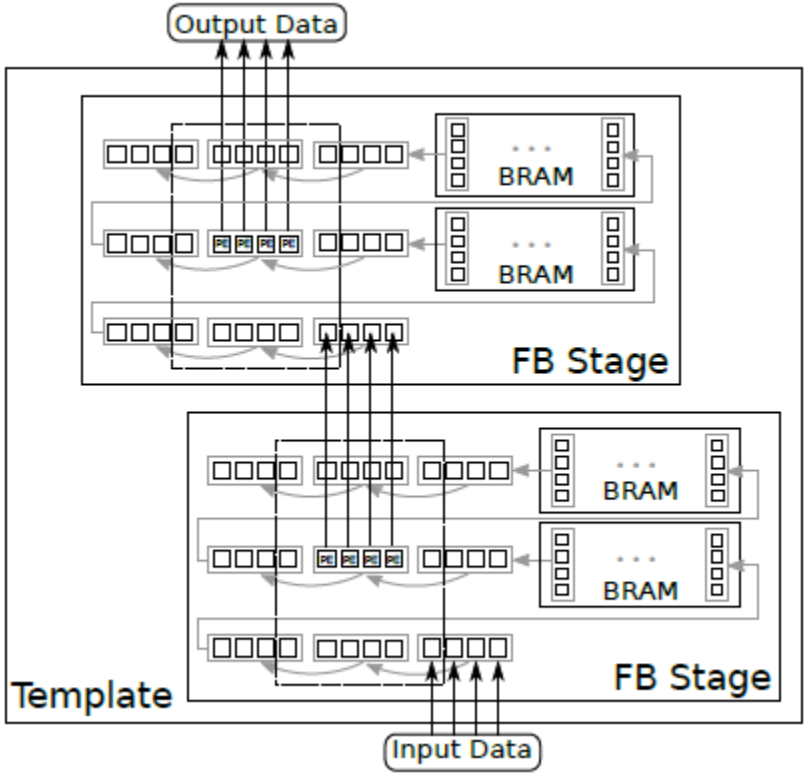
Verarbeitung von Stencils im Embedded-Umfeld



Verarbeitung von Stencils im Embedded-Umfeld

- Kontinuielricher Datenstom erlaubt Pipelining

  → Aufrollen der Zeitschleife



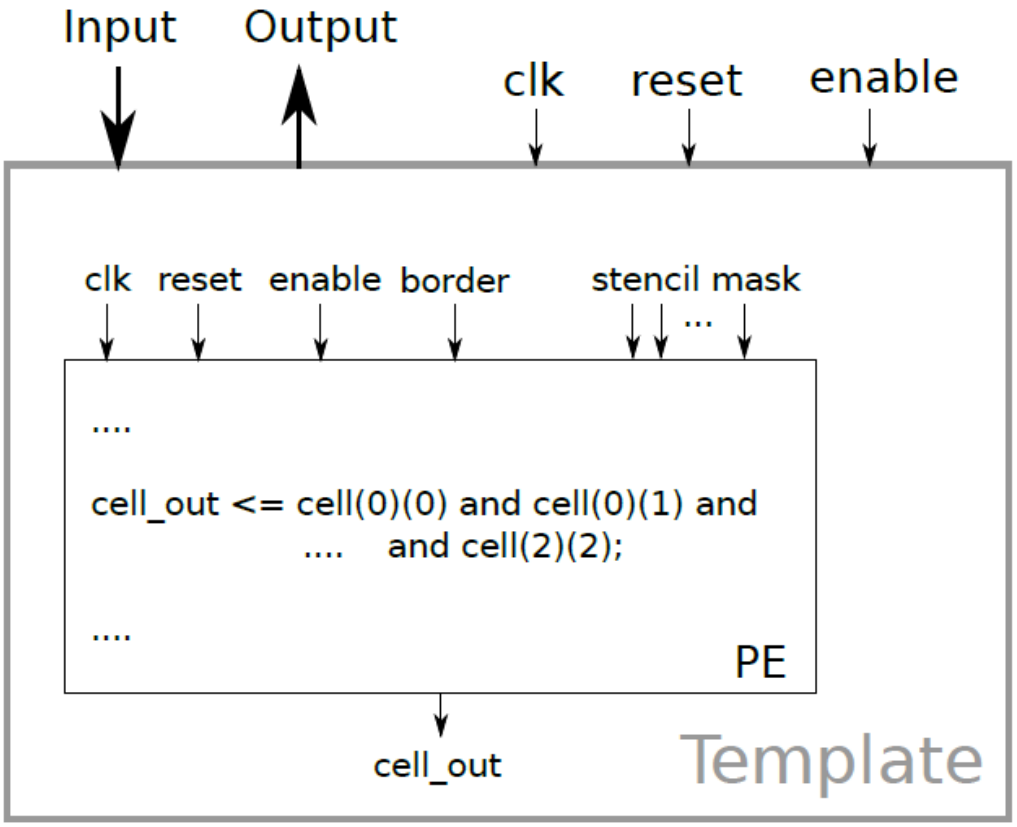

```
for t in TIME/2
   for s in SPACE/4+I
      parallel_exec
        // space unrolling
        update_cell(4*s+0,2*t+0)
         ...

         // time unrolling
        update_cell(4*s+0-I,2*t+1)
      ...
      end
   end
end
```

Verarbeitung von Stencils im Embedded-Umfeld

- Automatisierung der Synthese in Hardware
  Anwender gibt allgemeine Parameter vor
  - m × n – Auflösung Gitter
  - p × q – Größe Stencil
  - d - Bits pro Zelle
  - Par - Parallelisierung im Ort
  - it – Tiefe der Pipeline (Parallelisierung in Zeit)
  - tic – gewünschte Anzahl Zyklen in einem PE (arithmetische Pipeline)
- Verbindung Ein-/Ausgabe
- Implementierung Stencil Operation
- Implementierung Randbehandlung

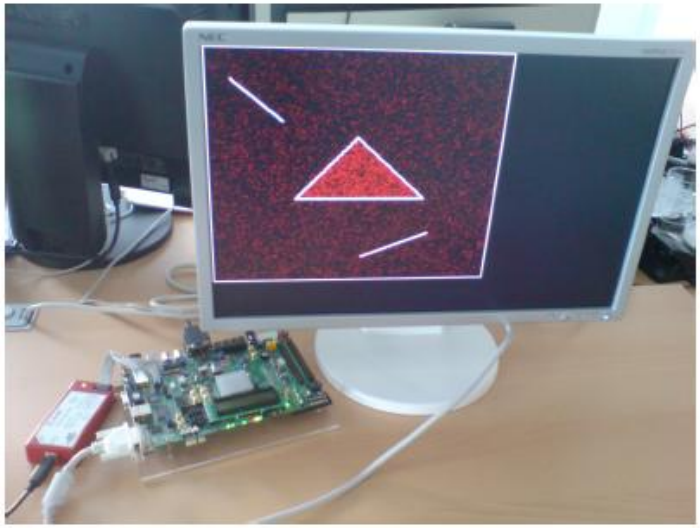
Verarbeitung von Stencils im Embedded-Umfeld

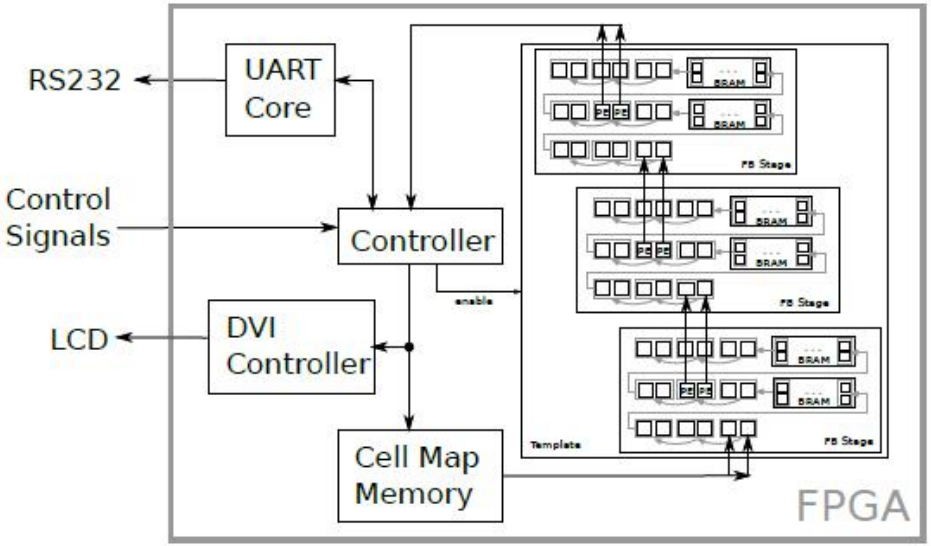
- Schnittstelle Template



Verarbeitung von Stencils im Embedded-Umfeld

- Schnittstelle Template





- Demonstrator on XUPV5 Board für Testzwecke
- I/O via UART
- Visualisierung auf einem LCD via DVI Schnittstelle

Verarbeitung von Stencils im Embedded-Umfeld

- Beispiel: MP Path Planning Implementierung
  - FPGA: Virtex-5 LX110T (XUPV5-Board)
  - Parameter: m = n = 1024, d = 5, 10000 Iterationen, 180 MHz

Tabelle: Synthesis Results

par	*it*	LUTs	Registers	BRAMs	$C_{total}(\times 10^8)$
1	1	0.52k	0.27k	1	104.86
8	1	1.13k	0.56k	3	13.11
16	1	1.83k	0.90k	5	6.55
8	8	6.93k	3.52k	24	1.64
8	16	13.60k	2.61k	48	0.82
8	32	26.56k	13.69k	96	0.41
8	64	62.37k	32.42k	96	0.20
16	16	24.04k	12.26k	80	0.41
16	32	49.63k	29.51k	80	0.20

Verarbeitung von Stencils im Embedded-Umfeld

- Leistungsvergleich
  - System Frequenz 180 MHz → 114 ms
  - GTX 480 Implementierung → 66 ms
  - → mit leistungsfähigerem FPGA, könnte auch mehr Leistung als mit GPU erzielt werden
  - leistungsfähigere GPU?
  - → bessere Effizienz bzgl. physikalischer Leistung
  - Embbeded Anwendung: Smarte Kamera

Zusammenfassung

- Bedeutung von Stencils für wissenschaftliches Rechnen für parallele eingebettete Hochleistungssysteme
- 1.Blick einfach – Optimierung im Detail kompliziert
- Ansätze für Automatisierung
  - Autotuning, DSL
- SW-Engineering-Ansatz über C++-Template
- Auotmatische Synthese über VHDL-Template
  - Beide Fällen User konzentriert sich auf Kernel

Danksagung

- LibGeoDecomp

  Andreas Schäfer + Vielzahl Studenten

- VHDL-Template

  Michael Schmidt, Marc Reichenbach + Studenten