Parallelisierung eines Lagrange'schen Partikelausbreitungsmodells

Abschlussbericht

Johann Weging, Cedrick Ansorge

6. Februar 2012

MPI- und OpenMP paralleles Modell zur Partikelbasierten Diffusionsmodellierung

1. sequentielles Modell
   - Besonderheiten
   - NetCDF-support
   - Datenaufteilung
   - Lastausgleich
   - `namelist`-reader
2. (paralleler Random-number generator)
3. MPI-Parallelisierung
4. OpenMP-Parallelisierung

Stochastische Integration⇒ Ensemble

Zeitliche Integration⇒Trajektorie

Meteorologischer Präprozessor

Bereitstellung der Parameter $\vec{v}(\vec{x})$, $\vec{\mu}(\vec{x})$, $\vec{K}(\vec{x})$

Umlagerung

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \Delta t \left(\frac{\partial \vec{K}}{\partial \vec{x}} + \vec{v}(x) \right) + 2\sqrt{\vec{K}} dW$$

Inkrementierung

eines Teilchenzählers $n_{\vec{x}}$am Ort $\vec{x}(t+\Delta t)$

Weitere Prozesse

- Auswaschung
- Umlagerung
- Deposition

⇒dynamisches g_p

Konzentrationsberechnung aus Teilchenzähler $C_{\Delta V} = \frac{g_p \sum_{\vec{x} \in \Delta V} n_{\vec{x}}}{T \Delta V}$

Stochastische Integration⇒ Ensemble

Zeitliche Integration⇒Trajektorie

Meteorologischer Präprozessor

=METRAS Modell (externes NetCDF File)

Umlagerung

$\vec{x}(t+\Delta t) = \vec{x}(t) + \Delta t \left(\frac{\partial \vec{K}}{\partial \vec{x}} + \vec{v}(x) \right) + 2\sqrt{\vec{K}} dW$

Inkrementierung

eines Teilchenzählers $n_{\vec{x}}$am Ort $\vec{x}(t+\Delta t)$

Weitere Prozesse

- keine

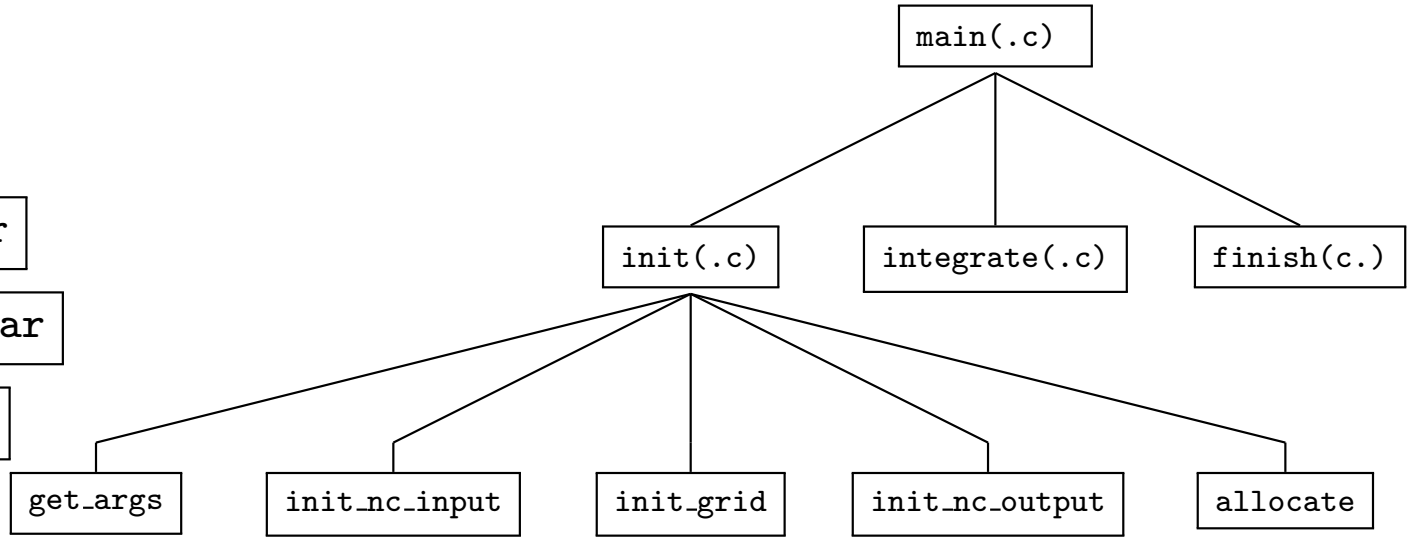
⇒ $g_p = const.s$

Konzentrationsberechnung aus Teilchenzähler $C_{\Delta V} = \frac{g_p \sum_{\vec{x} \in \Delta V} n_{\vec{x}}}{T \Delta V}$

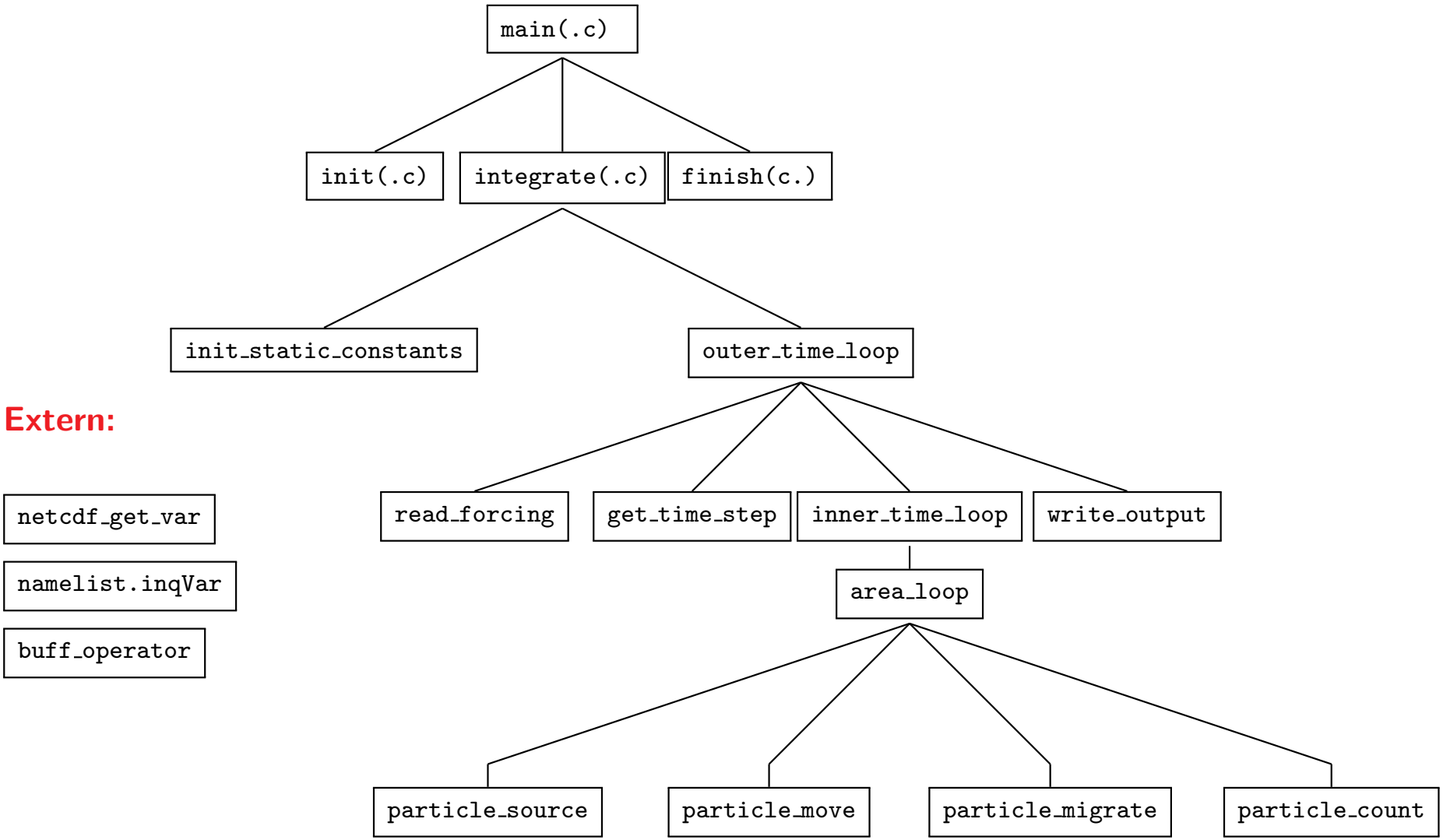
Ablaufplan (Init)

Extern:

- netcdf_get_var
- namelist.inqVar
- buff_operator



Ablaufplan (Integrate)



- Anordnung der Schleifen (stochastische/zeitlich)?

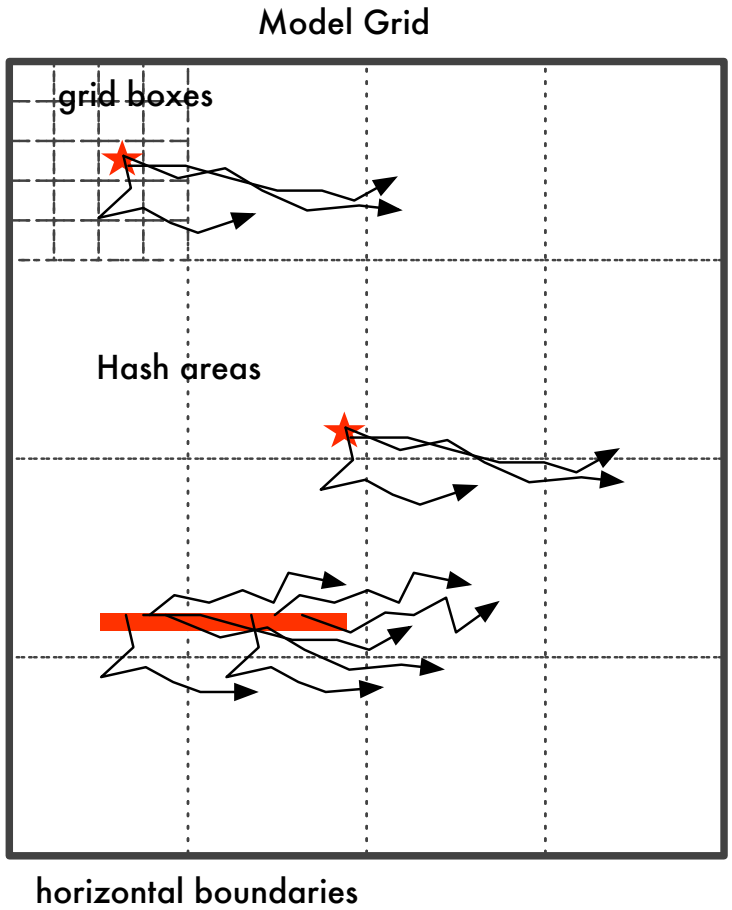
- Anordnung der Schleifen (stochastische/zeitlich)?
  - zeitliche Integration auen
  - Grund: zusammenhngende Leseaufrufe
- Bestimmung der Windgeschwindigkeit an beliebigen Orten?

- Anordnung der Schleifen (stochastische/zeitlich)?
  - zeitliche Integration auen
  - Grund: zusammenhngende Leseaufrufe
- Bestimmung der Windgeschwindigkeit an beliebigen Orten?
  - $\Rightarrow$ multi-(3D) lineare Interpolation
  - Herzstück des Modells $\approx 150 \frac{FloPs}{Partikel \cdot Zeitschritt}$
- Berücksichtigung von Gebäuden und Hindernissen implizit?

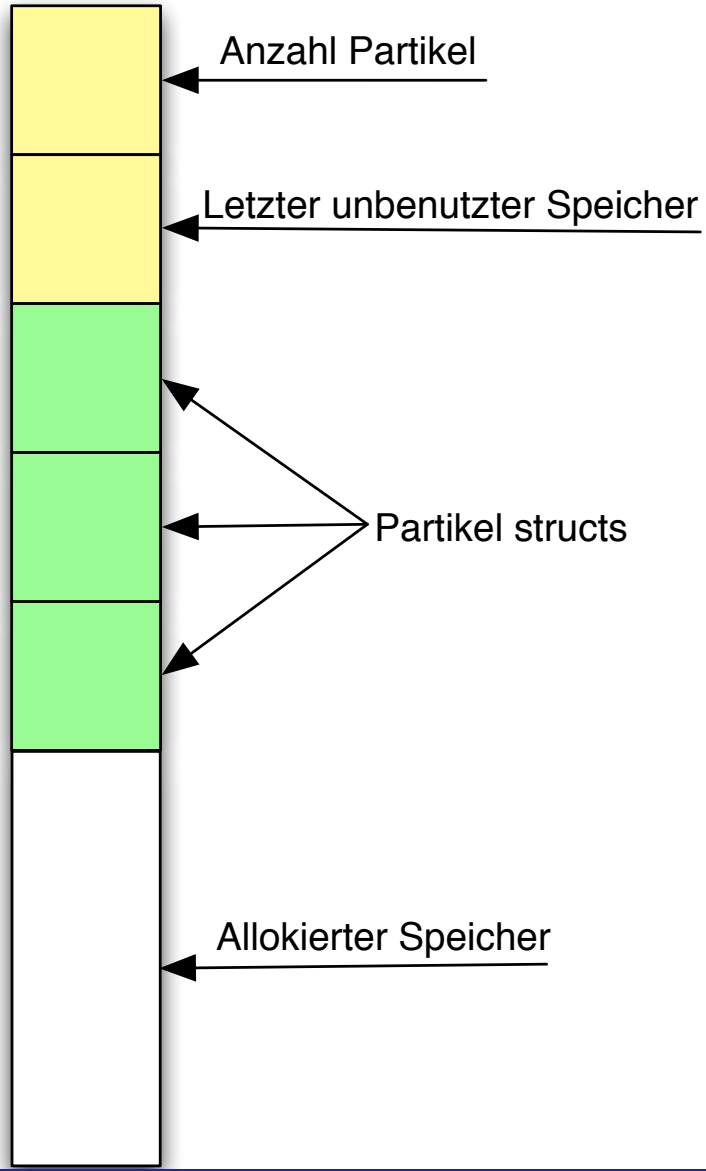
- Anordnung der Schleifen (stochastische/zeitlich)?
  - zeitliche Integration auen
  - Grund: zusammenhngende Leseaufrufe
- Bestimmung der Windgeschwindigkeit an beliebigen Orten?
  - $\Rightarrow$ multi-(3D) lineare Interpolation
  - Herzstück des Modells $\approx 150 \frac{FloPs}{Partikel \cdot Zeitschritt}$
- Berücksichtigung von Gebäuden und Hindernissen implizit?
  - setze $\mathbf{v}(\mathbf{x_{blocked}}) = \mathbf{v'^2}(\mathbf{x_{blocked}}) = 0$
  - gibt es erstmal nicht!
- flexible Gebietsgröße und Aufteilung

- Anordnung der Schleifen (stochastische/zeitlich)?
  - zeitliche Integration auen
  - Grund: zusammenhngende Leseaufrufe
- Bestimmung der Windgeschwindigkeit an beliebigen Orten?
  - $\Rightarrow$ multi-(3D) lineare Interpolation
  - Herzstück des Modells $\approx 150 \ \frac{FloPs}{Partikel \cdot Zeitschritt}$
- Berücksichtigung von Gebäuden und Hindernissen implizit?
  - setze $\mathbf{v}(\mathbf{x_{blocked}}) = \mathbf{v'^2}(\mathbf{x_{blocked}}) = 0$
  - gibt es erstmal nicht!
- flexible Gebietsgröße und Aufteilung
  - wird aus Dimensionen und (gefundenen) Koordinaten im Eingabefile bestimmt

- Zuordnung:
  Teilchen $\leftrightarrow$ **Gebiet** $\leftrightarrow$ Prozessor
- MPI-IO zum Einlesen von $\mathbf{v}(\mathbf{x}, \mathbf{t_0})$ $\quad, \mathbf{v'^2}(\mathbf{x}, \mathbf{t_0})$
- Übergabe von Teilchen bei Verlassen der Gebiete
  $\Rightarrow$ aufwendige Kommunikation
- Task-Konzept (...Lastausgleich):
  - definiere Gebiete
  - ordne Gebiete Prozessoren zu
  - (1) Integration, (2) interne Komm., (3) externe Komm.



Funktionsweise (=pointer magic :))



- Header zur besseren Verwaltung
- zusätzliches Element gibt größe des allokierten Speichers an, um realloc zu ermöglichen

MPI- und OpenMP paralleles Modell zur Partikelbasierten Diffusionsmodellierung

- ✓ sequentielles Modell
- (✓) Paralleles Modell
  - ✓ Besonderheiten
  - ✓ NetCDF-support
  - ✓ Datenaufteilung
  - ✓ Datenstruktur (ohne Defragmentierung)
  - ✓ `namelist-reader`
- (✓) MPI-Parallelisierung
- (✓) OpenMP-Parallelisierung - skaliert nicht

```
FOR - SCHLEIFE aeussere zeit-schleife
(Aktualisierung der meteorologischen Felder)

#BEGINN OMP Worksharing area
  WHILE - SCHLEIFE innere-zeit
  (zeitliche Integration)

    WHILE- SCHLEIFE Gebiete

       Partikel-Quelle
       Partikel-Verlagerung

         FOR- SCHLEIFE Partikel
           move every particle
         ENDE Partikel-Schleife
    ENDE Gebiets-Schleife

    WHILE- SCHLEIFE Gebiete
       Partikel-Migration
    ENDE Gebiets-schleife

  ENDE innere-zeite-Schleife

#END OMP Worksharing area

ENDE aeussere-zeit-Schleife
```

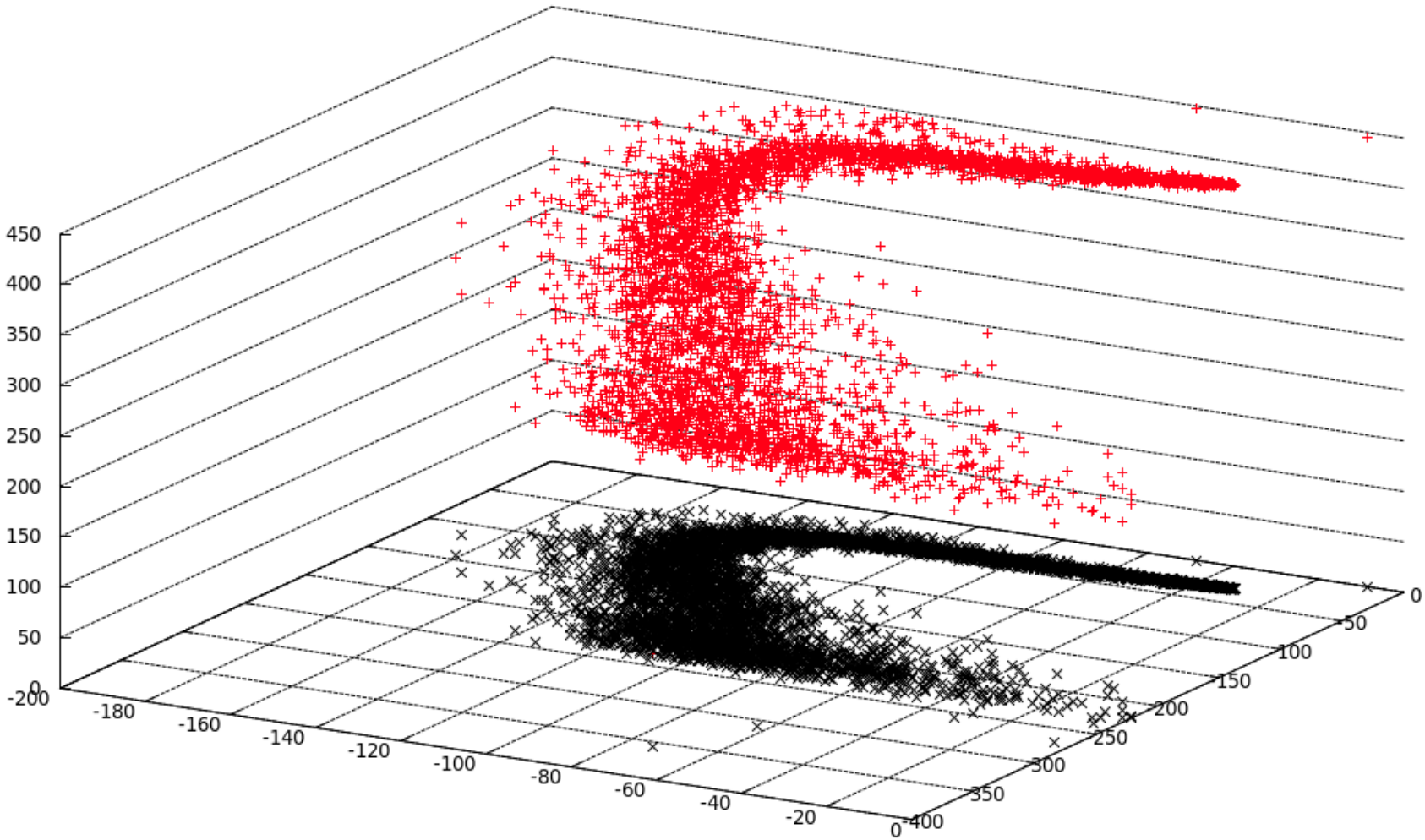
Skaliert nicht wegen unkoordinierten Speicherzugriffs

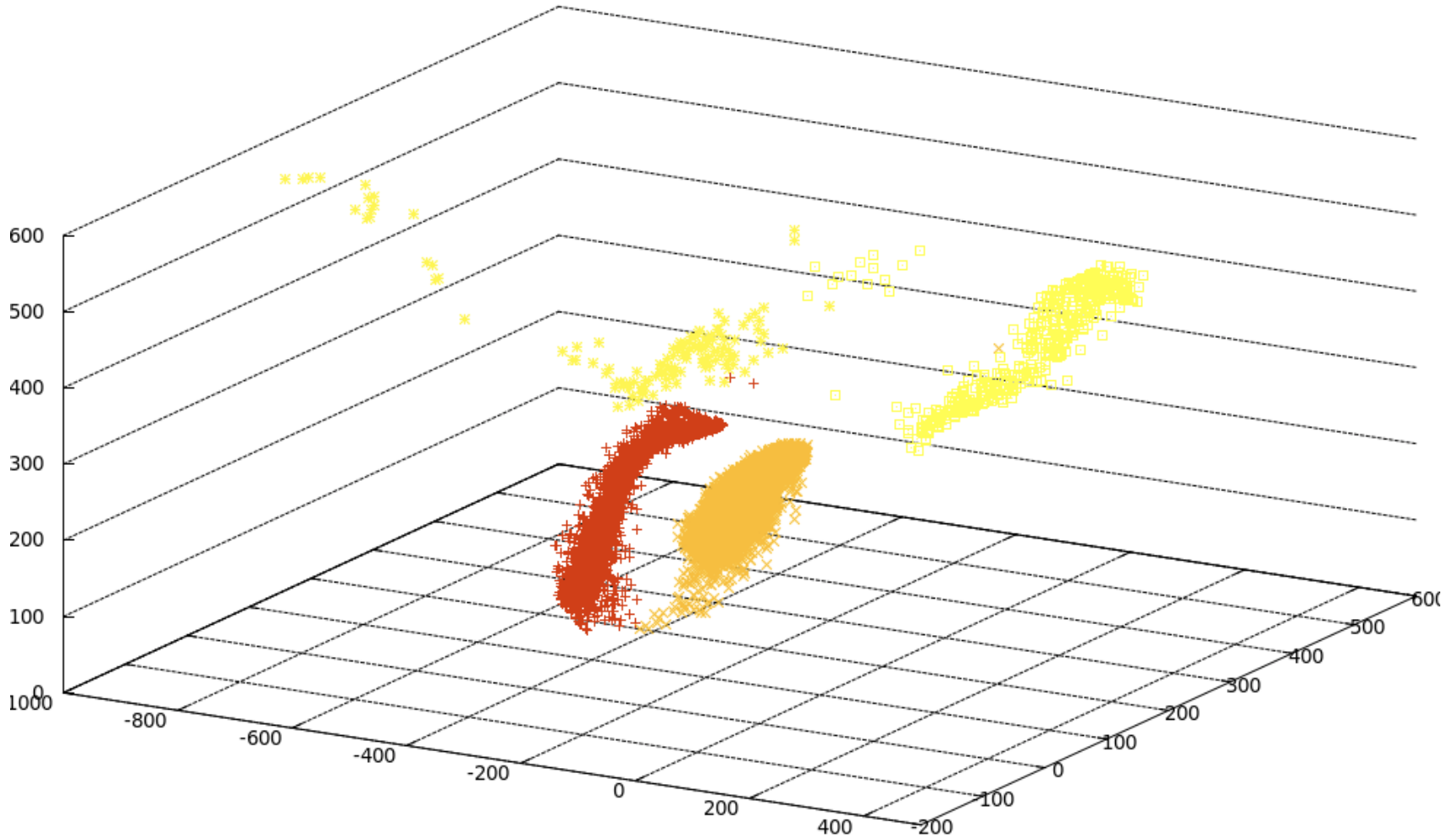
```
FOR - SCHLEIFE aeussere zeit-schleife
(Aktualisierung der meteorologischen Felder)

  FOR - SCHLEIFE innere-zeit
  (zeitliche Integration)

    FOR- SCHLEIFE Gebiete

       Partikel-Quelle
       Partikel-Verlagerung

#OMP PARALLEL for
         FOR- SCHLEIFE Partikel
           move every particle
         ENDE Partikel-schleife
#ENDE OMP PARALLEL FOR

    ENDE Gebiets-schleife

    FOR-Schleife Gebiete
      Partikel-Migration
    ENDE Gebiets-schleife

  ENDE innere-zeite-Schleife

ENDE aeussere-zeit-Schleife
```

Skaliert nicht wegen zu feiner Granularität der Parallelisierung

- Datenaufteilung nach Gebieten
  - Prozesse lesen nur daten für eigenes Gebiet (Dateien mehrfach geöffnet)
  - Output wird Knotenweise geschrieben → überlasse Parallelisierung dem FS
- globale Liste mit Area - Prozessor Zuordnung
- Verlagerung von Partikeln
  - sammle an andere Prozesse zu versendende Partikel während einer Iteration
  - `MPI_Isend`: verschicke *alle* zu versendenden Partikel an alle Prozesse, zu denen ein Partikel gesendet werden muss
  - `MPI_Iprobe`: überprüfe ob Nachrichten von irgendeinem Prozessor anliegen
  - `MPI_Irecv`: falls ja, empfange, und sortiere Partikel ein.

```
MPI_Iprobe(i, MPI_PARTICLE_LIST_TAG, MPI_COMM_WORLD,&flag1,&mpi_recvStatus1);
MPI_Iprobe(i, MPI_PARTICLE_WHERE_TAG,MPI_COMM_WORLD,&flag2,&mpi_recvStatus2);

if(flag1 && flag2) {
  struct Particle *p = (struct Particle *) calloc (1,sizeof(struct Particle) );

  MPI_Get_count(&mpi_recvStatus1,MPI_BYTE,&recvCount[0]);
  MPI_Get_count(&mpi_recvStatus2,MPI_INT,&recvCount[1]);

  MPI_Irecv(mpi_particle_bufInbox, recvCount[0], MPI_DOUBLE, i, MPI_PARTICLE_LIST_TAG,MPI_COMM_WORLD, sendListRequest);
  MPI_Irecv(mpi_migrateWhereIn, recvCount[1], MPI_INT, i, MPI_PARTICLE_WHERE_TAG,MPI_COMM_WORLD, sendWhereRequest);

  MPI_Wait(sendListRequest,MPI_STATUS_IGNORE);
  MPI_Wait(sendWhereRequest,MPI_STATUS_IGNORE);
}
```


450
400
350
300
250
200
150
100
50
0
-200
-180
-160
-140
-120
-100
-80
-60
-40
-20
0
-400
-350
-300
-250
-200
-150
-100
-50
0


600
500
400
300
200
100
0
1000
-800
-600
-400
-200
0
200
400
-200
-100
0
100
200
300
400
500
600