Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik die zukunft

Praktikum „Paralleles Programmieren“

Thema	Seiler Tide Modell
Autor	Anna Fuchs
Datum	Oktober 2012
Betreuer	Petra Nerge

Inhaltsverzeichnis

1. Vorwort

Im Rahmen des einsemestrigen Praktikums „Paralleles Programmieren“ soll ein vollständiges abgeschlossenes Programm mithilfe von MPI und ggf. anderen Methoden effizient parallelisiert werden.
In dieser Ausarbeitung wird das Seiler Tide Model (STM) parallelisiert, das zuerst nicht lauffähig mit älterem Code zur Verfügung gestellt wurde, dann von Petra Nerge[1] überarbeitet und zum Laufen gebracht wurde. Das Programm ist in Fortran 90/95 geschrieben und umfasst 12 Module.

Zum Beginn des Praktikums waren MPI und OpenMP Kenntnisse, aber keine Fortran Erfahrung vorhanden.

2. Einleitung

Erdsystemsimulation ist ein großes Forschungsgebiet und fest an Hochleistungsrechnen gebunden. Rechnen über große, aber auch kurze Zeiten führt sehr große Datenmengen und viel Rechenaufwand mit sich, so dass absehbare Ergebnisse nur mit Parallelrechnern erreicht werden können.
Ein parallelisiertes wissenschaftliches Modell vereint fachliche Kompetenzen aus mehreren Bereichen, wie der Physik, Mathematik und Informatik. Es gibt kaum einen Spezialisten, der im Stande wäre, alle diese Bereiche alleine zu beherrschen. Die Zusammenarbeit kann nur funktionieren, wenn gegenseitig die jeweils relevante Information zugänglich gemacht wird.
Diese Arbeit vermittelt die Sicht eines angehenden Informatikers und soll nicht die Mathematik und Physik der Anwendung erklären, sondern aus einer gegebenen Anwendung die nötigen Details für das Parallelisieren etablieren.
Es geht hierbei um eine vorgelegte Simulation der Mondgezeiten. Die wesentliche Rechenlast liegt dabei im numerischen Berechnen der Daten auf der Erdoberfläche.
Im folgenden Kapitel werden die Semantik der Anwendung und der relevante mathematische Hintergrund erklärt. Kapitel 4 und 5 erläutern die Datenabhängigkeiten und den Aufbau des Modells mit kommentierten Auszügen aus dem Originalcode. Im Anschluss daran wird im Kapitel 6 die Idee der Parallelisierung aufgeführt. Schwerpunkt dieser liegt bei der Vorbereitung der Kommunikation. Dafür soll eine virtuelle Topologie erstellt werden, die das Kommunizieren im wesentlichen erleichtern und transparenter machen soll. Am Ende des Kapitels wird eine Leistungsanalyse der erreichten Ergebnisse durchgeführt. Im Kapitel 7 werden dann Vorschläge zur Verbesserung der Last mit entsprechendem Code in C vorgeführt. Nach der Zusammenfassung im Kapitel 8 befindet sich der Anhang mit relevanten technischen Hintergrundinformationen zu der Arbeit und anschließendem Literaturverzeichnis im Kapitel 10.

1 http://wr.informatik.uni-hamburg.de/people/petra_nerge

3. Anwendung

3.1 Gezeiten

Seiler Tide Model (STM) ist ein Modell zum Simulieren der globalen Ozeangezeiten. Unter Gezeiten versteht man Massenbewegungen der Erde, die aus den Anziehungskräften der Erde, des Mondes, der Sonne und der anderen Himmelskörper und aus der Erdrotation resultieren. Auf den Mond ist der überwiegende Anteil an den Gezeiten auf der Erde zurückzuführen.
Die Mondbahn kann als Ellipse überlagert mit Störungen von den anderen Himmelskörpern beschrieben werden. Dies kann näherungsweise mit periodischen Termen dargestellt werden. Aus periodischen Kräften resultieren periodische Massenbewegungen. In diesem Modell geht es um die Mondgezeiten im halbtägigen Zyklus. Schematisch kann die Wechselwirkung so dargestellt werden:

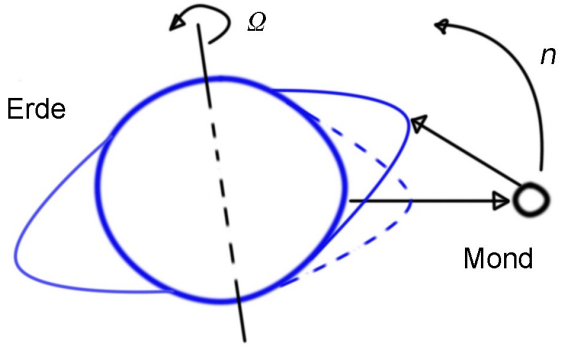


Abb. 1.a Schematische Darstelung der Entstehung von Flutbergen

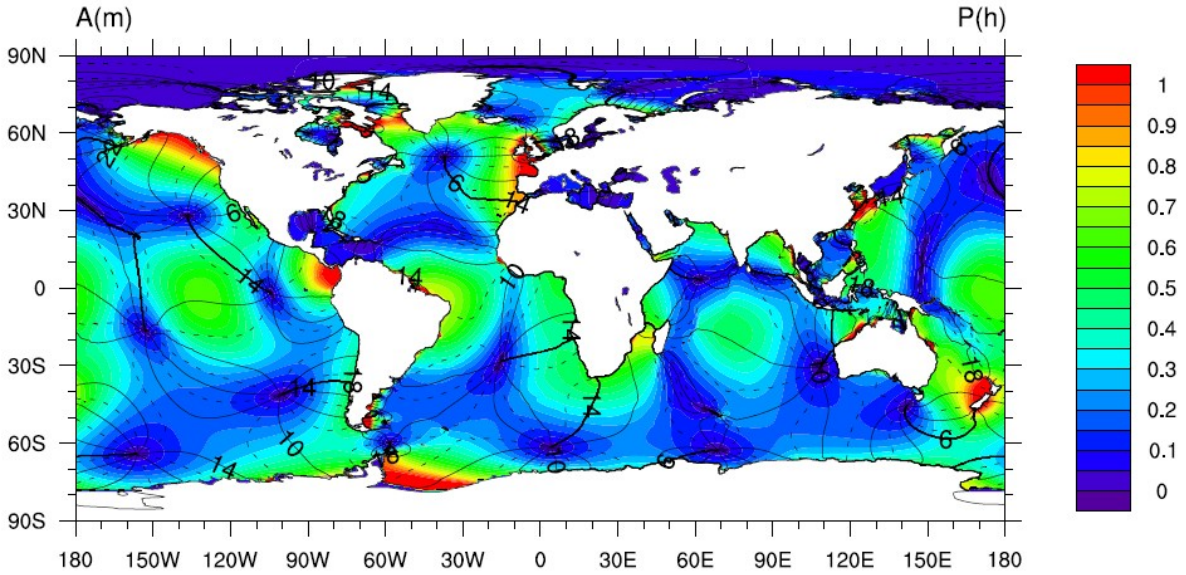


Abb. 1.b Vom Max Planck Institut berechnete Wellenhöhen in m

Aus der gegenseitigen Anziehung von Erde und Mond sowie der Erdrotation entstehen z.B. im Ozean Wasserberge. Die Erdrotation Ω ist höher, als die Umlaufsgeschwindigkeit des Mondes *n* und die eben entstandenen Wasserberge laufen dem Mond vor und wechselwirken gravitativ mit dem Mond. Der Energieaustausch bewirkt u.A. die Beschleunigung des Mondes und Verlangsamung der Erdrotation. Direkte Konsequenz daraus ist z.B. die Veränderung der Tageslänge. Die Gezeiten bewirken unregelmäßige Schwankungen bis zu 0,005 *s* jährlich, periodische Schwankungen bis zu 0,001 *s* monatich, dauerhafte Störung etwa +0,0016 *s* pro Jahrhundert (Wenzel, 1995). Darüber hinaus wirken sich die Kräfte auch auf die Bewegung der Drehachse (etwa +/-100 *nrad*) und deformieren den Erdkörper (um +/-0,3 *m*). Für die dauerhafte Störung der Erdrotation zeigen paläontologische Daten eine Abnahme der heutigen Tageslänge von 24 Stunden auf 21,9 Stunden vor 650 Mio. Jahren (Williams, 2000).
Für das Modell wichtigste Auswirkungen sind jedoch Ozeangezeiten (Resonanz der Gezeitenwelle in Ozeanen und Schelfgebieten[2] um +/- 0,5 *m* bis +/- 5 *m*).
Die Erkenntnisse über Gezeiten sind sehr wichtig in der Wissenschaft (Ozeanografie, Astronomie, Geophysik etc.) aber auch in der Schifffahrt. Mit dem nötigen Wissen kann man u.A. viel Treibstoff sparen, wenn man die geeigneten Routen abhängig von den aktuellen Gezeiten auswählt.

Es spielen viele Faktoren eine Rolle bei der Berechnung der Gezeiten. Mit der Entwicklung nach Kugelflächenfunktionen kann das Gezeiten erzeugende Potential[3] eines Himmelskörpers *i* auf der Erde beschrieben werden:

$$\Phi_i = \frac{\gamma \cdot M_i}{d_i} \cdot \sum_{l=2}^{\infty} \left(\frac{r}{d_i}\right)^l \cdot \sum_{m=0}^{l} (2-\delta_{om}) \cdot \frac{(l-m)!}{(l+m)!} \cdot P_l^m \cdot (\sin\varphi) P_l^m \cdot (\sin\delta_i) \cdot \cos m \cdot (\lambda + \tau_i)$$

wobei

γ *Gravitationskonstante*
M_i *Masse*
d_i *Abstand zwischen Erdmittelpunkt und dem Massenschwerpunkt des Himmelskörpers i*
δ_i *Deklination des Himmelskörpers i*
τ_i *Stundenwinkel des Himmelskörpers i*
λ *geographische Länge*
φ *geographische Breite*
P *Legendrische Funktionen* 1. *Art vom Grad l und Ordnung m*
δ_{om} $1\ für\ m=0\ \ und\ \ 0\ sonst$

bezeichnen.

Das *m* kann variabel eingestellt werden:
m = 0 : konstant und langperiodisch mit einer Periode von 14 Tagen bis 18,6 Jahren
m = 1 : ganztägig mit einer Periode von etwa 24 Stunden
m = 2 : halbtägig mit einer Periode von etwa 12 Stunden

Das Gezeiten erzeugende Potential auf der Erde kann in guter Näherung mit der 2. Ordnung beschrieben werden. Die Terme der Reihenentwicklung werden als Partialtiden bezeichnet.

3.2 Ozeangezeiten

Das Bewegungsfeld des Ozeans wird vollständig durch das nichtlineare Differentialgleichungssystem der *Navier-Stokes-Gleichungen*[4] beschrieben. Bei Betrachtung der Ozeangezeiten kann die vertikale Geschwindigkeit gegenüber den horizontalen Geschwindigkeiten vernachlässigt und die Druckverteilung als hydrostatisch beschrieben werden. In sphärischer Polarnotation relativ zum Meeresboden haben dann

2 Flacher, küstennaher Boden bis zu 200 m unter dem Meeresspiegel - http://de.wikipedia.org/wiki/Schelf
3 http://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&cad=rja&ved=0CFIQFjAG&url=http%3A%2F%2Fwww.physik.unibe.ch%2Funibe%2Fphilnat%2Ffachbphysik%2Fcontent%2Fe4897%2Fe4911%2Fe5671%2Fe6804%2Fe6806%2Ffiles7037%2F21_koordinatenschaetzung_h_ger.pdf&ei=kUZvUL_IFtHAtAb3zICQBQ&usg=AFQjCNFAK5pR7n3lvRXBsQ8xI42FLijTCQ&sig2=3bMrcxFtekeI9d8gfZKlqQ
4 http://de.wikipedia.org/wiki/Navier-Stokes-Gleichungen

die Gleichungen folgende Form:

$$\frac{\partial u}{\partial t} + \frac{u}{R \cdot cos\varphi}\frac{\partial u}{\partial \lambda} + \frac{v}{R}\frac{\partial u}{\partial \varphi} - 2\Omega sin(\varphi)v = -\frac{g}{R \cdot cos\varphi}\frac{\partial \zeta}{\partial \lambda} + \frac{1}{R \cdot cos\varphi}\frac{\partial \Phi_j^r}{\partial \lambda} + \Gamma_{B,\lambda} + \Gamma_{E,\lambda}$$

$$\frac{\partial v}{\partial t} + \frac{u}{R \cdot cos\varphi}\frac{\partial v}{\partial \lambda} + \frac{v}{R}\frac{\partial v}{\partial \varphi} - 2\Omega sin(\varphi)u = -\frac{g}{R}\frac{\partial \zeta}{\partial \varphi} + \frac{1}{R}\frac{\partial \Phi_j^r}{\partial \varphi} + \Gamma_{B,\varphi} + \Gamma_{E,\varphi}$$

$$\frac{\partial \zeta}{\partial \varphi} = -\frac{1}{Rcos\varphi}\left[\frac{\partial}{\mathrm{d}\lambda}(H \cdot u) + \frac{\partial}{\partial \varphi}(H \cdot v \cdot cos\varphi)\right]$$

wobei

Φ_j^r *Term der Reihenentwicklung des Gezeiten erzeugenden Potentials durch den Himmelskörper j*

u bzw. v vertikal gemittelte zonale bzw. meridionale Geschwindigkeit

ζ *Wasserstand*

R Erdradius

H ungestörte Wassertiefe + Wasserstand

$\Gamma_{B,\lambda}, \Gamma_{B,\phi}$ *Bodenreibung*

$\Gamma_{E,\lambda}, \Gamma_{E,\phi}$ *horizontale Impulsdiffusion*

bezeichnen.

Die Gleichungen beinhalten die Wechselwirkungen aus der Erdrotation, dem Druckgradienten, der Advektion, der Reibung und der Diffusion.
Ausführliche Informationen über die Anwendung können u. A. in Seiler (1989) und Nerge (1998) nachgelesen werden.

Es gibt grundsätzlich zwei Wege ein solches Gleichungssystem zu lösen: Linearisierung und numerische Lösung. Bei der Linearisierung geht es um das Umwandeln nichtlinearer Differentialgleichungen in lineare zum Zwecke der Anwendung gewöhnlicher Lösungsverfahren. Mit einer Näherung entfernt man sich jedoch immer mehr vom ursprünglichen Problem. Da es sich hier um ein nichtlineares Anfangswerteproblem handelt, kommt nur die numerische Lösung in Frage.
Des Weiteren ist die Lösungsmenge einer Differentialgleichung im Allgemeinen nicht durch die Gleichung selbst eindeutig bestimmt, sondern benötigt zusätzlich noch die Anfangs- und Randwerte.[5]

5 http://de.wikipedia.org/wiki/Differentialgleichung

4. Seiler Tide Model (STM)

Das Modell wurde von Ulrike Seiler 1989 – Institut für Meereskunde der Universität Hamburg – für Simulationen der wesentlichen halb-, eintägigen und langperiodischen Partialtiden zur Untersuchung der instantanen Drehimpuls- und Energiebilanzierung entwickelt (Seiler, 1989).

Die numerische Lösung der Gleichungen erfolgt mit der Gitterpunktmethode. Das Bewegungsfeld ist dabei nach der finiten Differenzen Methode[6] ausgedrückt. Das Gitternetz über dem Globus besteht aus Längen (*longitudes*) und Breiten (*latitudes*) (Abb. 2)[7]. Die Auflösung beträgt 1°. Die Abstände zwischen den Längen nehmen zu den Polen ab. Für die Berechnung wird das Gitter auf ein rechteckiges Gitter projiziert (Abb. 3).

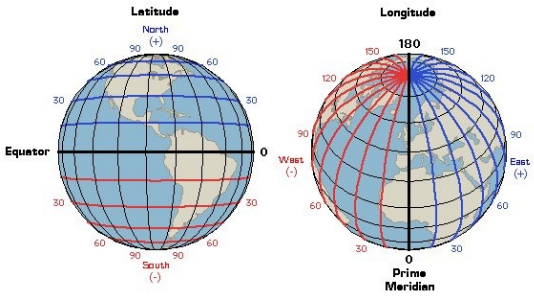


Abb. 2

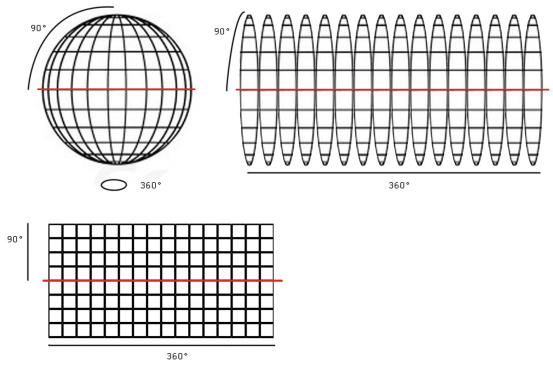


Abb. 3 Schematische Projektion

An den Gitterpunkten werden die Variablen definiert. Hierzu wird ein Arakawa-C Gitter[8] verwendet. Im Gegensatz zum z. B. A-Gitter, bei dem alle Variablen am gleichen Gitterpunkt definiert sind, liefert die Berechnung auf dem C-Gitter viel genauere Ergebnisse. Vertikal und horizontal werden zentrierte Differenzenquotienten verwendet (Mittelwerte aus benachbarten Gitterzellen gebildet). Durch Halbierung der Maschenweite wird der Fehler um den Faktor vier kleiner.
In der Abbildung 4 ist das Arakawa-C Gitter dargestellt. $\Delta\varphi$ bzw. $\Delta\lambda$ bezeichnet den Winkelabstand der Wasserstandpunkte ζ und der Geschwindigkeitspunkte *u*, *v* in der geographischen Breite bzw. Länge. Die Tiefenverteilung D wird an den ζ - Punkten vorgegeben. D^u, D^v stellen als arithmetischen Mittelwert der benachbarten Tiefen *D* und die Tiefe an den *u*- und *v*-Punkten dar.

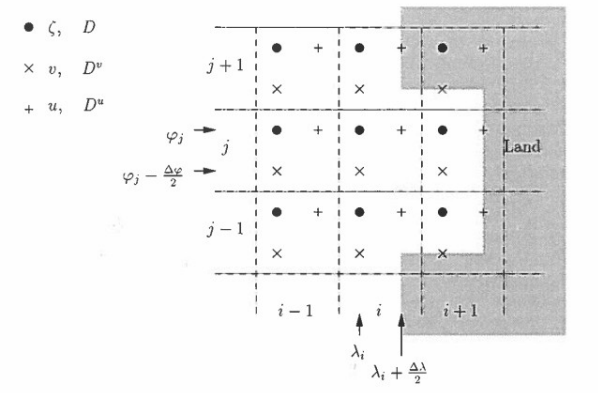


Abb. 4 Belegung der Wasserstands- und Geschwindigkeitspunkte im Arakawa-C Gitter

6 http://de.wikipedia.org/wiki/Finite-Differenzen-Methode
7 http://www.geographyalltheway.com/ks3_geography/maps_atlases/imagesetc/latitudelongitude.jpg
8 http://en.wikipedia.org/wiki/Arakawa_grids

Um das oben genannte nichtlineare Differentialgleichungssystem zu lösen, wird in diesem Modell ein semiiplizites Rechenverfahren verwendet. Die Gleichung für den Wasserstand wird implizit, die Gleichungen der Geschwindigkeiten werden explizit gelöst, wobei der Druckgradient sowohl explizit als auch implizit betrachtet wird.

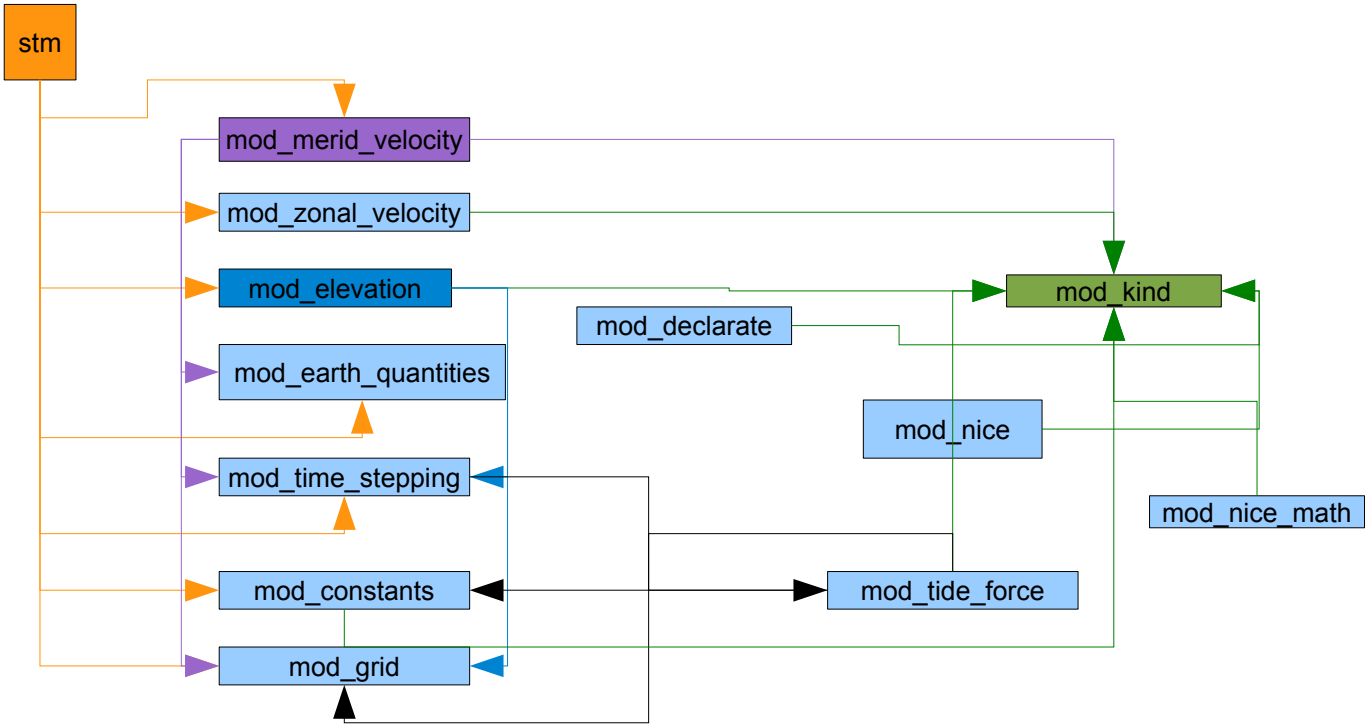
Beim expliziten Verfahren werden die Gleichungen für den nächsten Zeitschritt aus den Werten zum gegenwärtigen Zeitschritt berechnet. Beim impliziten Verfahren werden die Gleichungen für den nächsten Zeitschritt iterativ aus genäherten Anfangswerten für den nächsten Zeitschritt gelöst. Diese genäherten Anfangswerte sind oft Erfahrungswerte. In diesem Modell heißt es, dass der arithmetische Mittelwert aus Größen zum gegenwärtigen und nächsten Zeitpunkt berechnet wird. Hinsichtlich des Wasserstandes ζ wird über den gesamten Globus iteriert, Nord- und Südhalbkugel abwechselnd. Die nördliche Halbkugel unterscheidet sich jedoch darin, dass die Nordpolarkappe (4° Bereich am Nordpol, Wert einstellbar) getrennt berechnet wird.

Die Mitteilungen im impliziten Verfahren und die Diskretisierung führen letztlich zu einer Massenimbalance. Das Modell sieht hierfür ein Abbruchkriterium für das iterative Verfahren vor, das den Grad der Massenerhaltung bestimmt.

Das Verständnis dieses Gitters und des Aufbaus ist wichtig für die Bestimmung der Datenabhängigkeiten.

5. Implementierung

5.1 Moduldiagramm



5.2 Zeitanalyse

Die ersten Zeitanalysen erfolgen mit *gprof*. Das Programm wird dabei nicht mit einer geladenen Topographie, sondern mit Testdaten, die einer vollständig mit Wasser bedeckten Erde entspricht , ausgeführt (siehe mehr im Kapitel Lastausgleich). Dabei läuft die sequentiellen Ausführung 1min44 (mit *bash command time* gemessen) und liefert folgenden Analyseverlauf (gekürzt):

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 83.21     83.02    83.02     2001     0.04     0.04  __mod_elevation_MOD_elevation_iteration_relaxation
  1.81     84.83     1.81        1     1.81    99.79  MAIN__
  1.77     86.60     1.77     2001     0.00     0.00  __mod_elevation_MOD_elevation_iteration_matrix_prepare
  1.01     87.61     1.01     2001     0.00     0.00  __mod_elevation_MOD_elevation_iteration_zeta_coefficient
  1.01     88.62     1.01 126783360     0.00     0.00  __mod_merid_velocity_MOD_merid_vel_bottom_friction
  0.92     89.54     0.92 126783360     0.00     0.00  __mod_zonal_velocity_MOD_zonal_vel_bottom_friction
  0.87     90.41     0.87     2001     0.00     0.00  __mod_merid_velocity_MOD_merid_vel_advection
  0.76     91.17     0.76     2001     0.00     0.00  __mod_zonal_velocity_MOD_zonal_vel_advection
...
```

Es zeigt sich eindeutig, dass primär die Funktion **`iteration_relaxation`** (implizite Lösung der Wasserstandsgleichung nach dem *„Successive Over-Relaxation“-Verfahren*[9]) parallelisiert werden sollte. Aufgrund des Amdahls Gesetzes[10] wäre selbst die optimale Parallelisierung anderer Funktionen im Vergleich zum Zeitverbrauch der erwähnten Funktion als gering anzusehen.
Die Routine befindet sich im Modul *mod_elevation.f90.*

9 http://de.wikipedia.org/wiki/SOR-Verfahren
10 http://de.wikipedia.org/wiki/Amdahlsches_Gesetz

Data Types

```
interface  elevation_iteration_zeta_coefficien
```

Public Member Functions

```
  pure real(kind=wp)
           function,
dimension(longs_star
       t:longs_end,
lats_start_zeta:lats elevation_iteration_zeta_coefficient (u_new_exp, u_last,
         _end_zeta)  total_depth_u_points, v_new_exp, v_last, total_depth_v_points)

    pure subroutine  elevation_iteration_matrix_prepare (coefficient_east, coefficient_west,
                     coefficient_south, coefficient_north, coefficient_zeta, u_new_exp, u_last,
                     v_new_exp, v_last, total_depth_u_points, total_depth_v_points,
                     zonal_pressure_gradient_coef_imp, merid_pressure_gradient_coef_imp)

    pure subroutine  elevation_iteration_relaxation (zeta_new, coefficient_east,
                     coefficient_west, coefficient_south, coefficient_north, coefficient_zeta,
                     residuum_max, count_checks, stat)
```

Die entscheidende Schleife in dieser Routine:

```
1   !--- start iteration
2   DO count_checks = 0 , max_convergence_checks                                                    Konvergenziterationen
3      DO count_iteration = 1 , number_iterations
4      !--- set southerly hemisphere to iterate first
5      hemisphere = 1                                                                               Parameter für Südhalbkugel
6      start_lat  = start_south_lat + 1
7      end_lat    = end_south_lat
8         DO WHILE ( hemisphere < 3 )
9         !--- latitudes without boundaries                                                        Berechnung des residuums, die
10           DO lat = start_lat , end_lat                                                         es in erster Linie gilt zu
11              DO lon = longs_start , longs_end                                                  parallelisieren
12                 residuum         (lon , lat)                          = &
13                    coefficient_north(lon , lat) * zeta_new(lon    , lat+1) + &                Koeffizienten werden nur
14                    coefficient_south(lon , lat) * zeta_new(lon    , lat-1) + &                ausgelesen, wurden schon
15                    coefficient_east (lon , lat) * zeta_new(lon+1 , lat  ) + &                 berechnet
16                    coefficient_west (lon , lat) * zeta_new(lon-1 , lat  ) - &
17                    zeta_new(lon    , lat  ) + &
18                    coefficient_zeta (lon , lat)
19             END DO
20          END DO
21       !--- set northerly hemisphere to iterate next                                            Wechsel zur nördlichen
22       hemisphere = hemisphere + 1                                                              Halbkugel
23       start_lat  = start_north_lat
24       end_lat    = end_north_lat - 1
25    END DO
26
27    !--- boundary condition for southernmost latitude                                           Berechnung des residuum 's
28    DO lon = longs_start , longs_end
29       residuum         (lon , start_south_lat)                                    = &         Auch hier sind die nötigen
30          coefficient_north(lon , start_south_lat) * zeta_new(lon    , start_south_lat+1) + &  Koeffizienten schon berechnet
31          coefficient_east (lon , start_south_lat) * zeta_new(lon+1 , start_south_lat  ) + &
32          coefficient_west (lon , start_south_lat) * zeta_new(lon-1 , start_south_lat  ) - &
33          zeta_new(lon    , start_south_lat  ) + &
34          coefficient_zeta (lon , start_south_lat)
35    END DO
36
37    !--- boundary condition for northernmost latitude
38    residuum_polar = coefficient_zeta(longs_start , lats_end_zeta)                             Berechnung der Pole
39    DO lon = longs_start , longs_end
40       residuum_polar = residuum_polar                    &
41          + coefficient_south(lon , lats_end_v) &
42          *          zeta_new(lon , lats_end_u)
43    END DO
44    residuum(longs_start:longs_end , lats_end_zeta) = &
45    residuum_polar                                  - &
46    zeta_new(longs_start:longs_end , lats_end_zeta)
47
48    !--- cyclic boundary condition for longitudes on southerly hemisphere                       Boundary Conditions
49    residuum(longs_start_sub_1 , start_south_lat:end_south_lat) = &
50    residuum(longs_end         , start_south_lat:end_south_lat)
51
52    residuum(longs_end_add_1   , start_south_lat:end_south_lat) = &
53    residuum(longs_start       , start_south_lat:end_south_lat)
54
55    !--- cyclic boundary condition for longitudes on northerly hemisphere
56    residuum(longs_start_sub_1 , start_north_lat:end_north_lat) = &
57    residuum(longs_end         , start_north_lat:end_north_lat)
58
59    residuum(longs_end_add_1   , start_north_lat:end_north_lat) = &
60    residuum(longs_start       , start_north_lat:end_north_lat)
61
```

```
62      !--- new zeta on southerly hemisphere
63      zeta_new(longs_start_sub_1:longs_end_add_1 , start_south_lat:end_south_lat) = &
64      zeta_new(longs_start_sub_1:longs_end_add_1 , start_south_lat:end_south_lat) + &
65      residuum(longs_start_sub_1:longs_end_add_1 , start_south_lat:end_south_lat)
66
67      !--- new zeta on northerly hemisphere
68      zeta_new(longs_start_sub_1:longs_end_add_1 , start_north_lat:end_north_lat) = &
69      zeta_new(longs_start_sub_1:longs_end_add_1 , start_north_lat:end_north_lat) + &
70      residuum(longs_start_sub_1:longs_end_add_1 , start_north_lat:end_north_lat)
71      END DO
72
73      !--- check convergence determination                                          Abbruchkriterium Abfrage
74      IF ( max_convergence_checks < 1 ) THEN
75         stat = .true.
76         RETURN
77      END IF
78
79      !--- initialize maximum of residuum to false
80      residuum_max = -99999._wp
81
82      !--- set maximum of residuum to false for southern hemisphere
83      residuum_south_max = -99999._wp
84      !--- determine area on southern hemisphere where no further iteration is necessary
85      !    set end latitude of next iteration for southerly hemisphere
86      DO end_south_lat = end_south_lat , start_south_lat , -1
87         DO lon = longs_start , longs_end
88            residuum_south_max = MAX( residuum_south_max                  , &       residuum vergleichen für
89                                      ABS( residuum(lon , end_south_lat) )   &       Südhalbkugel
90                                      )
91         END DO
92         IF ( residuum_south_max > convergence_criterion) EXIT
93            residuum_max = MAX( residuum_max , residuum_south_max )
94      END DO
95      IF ( end_south_lat >= start_south_lat .and. end_south_lat < lats_equator_u ) THEN
96          end_south_lat = end_south_lat + 1
97      END IF
98
99      !--- set maximum of residuum to false for northern hemisphere
100     residuum_north_max = -99999._wp
101     !--- determine area on northern hemisphere where no further iteration is necessary   residuum vergleichen für
102     !    set start latitude of next iteration for northerly hemisphere                  Nordhalbkugel
103     DO start_north_lat = start_north_lat , end_north_lat
104        DO lon = longs_start , longs_end
105           residuum_north_max = MAX( residuum_north_max                    , &
106                                     ABS( residuum(lon , start_north_lat) )   &
107                                      )
108        END DO
109        IF ( residuum_north_max > convergence_criterion) EXIT
110        residuum_max = MAX( residuum_max , residuum_north_max)
111     END DO
112     IF ( start_north_lat > lats_equator_v .and. start_north_lat <= end_north_lat ) THEN   Abbruch wenn Konvergenz
113        start_north_lat = start_north_lat - 1                                            hinreichend
114     !--- check convergence on both hemispheres
115     ELSE IF ( start_north_lat > end_north_lat .and. end_south_lat < start_south_lat ) THEN
116             stat =.true.
117             RETURN                                                                     Erst hier endet die Schleife
118     END IF
119  END DO
120
121
123
124
```

6. Parallelisierung

6.1 Topologie und Gebietszerlegung

In erster Linie soll eine virtuelle Topologie aufgestellt werden, die bequemere und übersichtlichere Kommunikation ermöglicht. Die Erdhalbkugeln werden für den Wasserstand abwechselnd berechnet. Es ist optimal in jeder Iteration die gesamte Prozesszahl auszunutzen. Da die Halbkugeln bis auf die Nordpolarkappe identisch aufgebaut sind, lässt sich eine Topologie aufstellen, die einem 360x90 Gitter entspricht. Entlang der Längen wird periodisch gerechnet, es entsteht also ein Ring. Entlang der Breiten wird vom Äquator (0°) bis 86°N auf der Nordhalbkugel und auf der Südhalbkugel bis 90°S gerechnet. Es bietet sich also eine Zylindertopologie an, die gespiegelt bei jeder Iteration verwendet wird. Dabei ist die getrennte Berechnung der Nordpolarkappe und der am Äquator benötigten Werte zu berücksichtigen.

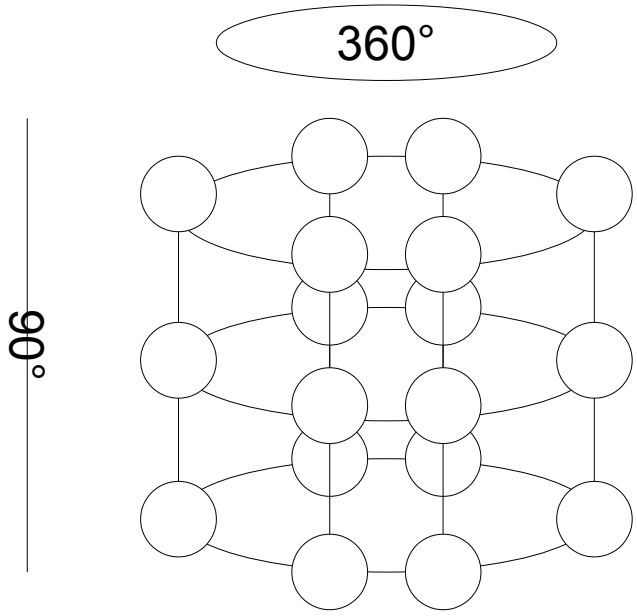


Abb. 5 Zylindertopologie

MPI Initialisierung:

```
!MPI
INTEGER :: IERROR
CALL MPI_INIT(IERROR)
```

Um eine virtuelle Topologie zu erzeugen wird folgende Funktion verwendet[11]:

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims, int *periods,
                    int reorder, MPI_Comm *comm_cart)

Input Parameters

comm_old
      input communicator (handle)
ndims
      number of dimensions of cartesian grid (integer)
dims
      integer array of size ndims specifying the number of processes in each dimension
periods
      logical array of size ndims specifying whether the grid is periodic (true) or not (false) in
      each dimension
```

11 http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Cart_create.html

reorder
ranking may be reordered (true) or not (false) (logical)

Output Parameter

comm_cart
communicator with new cartesian topology (handle)

```
CALL MPI_Cart_create(old_comm, ndims, dim_size, periods, reorder, cylinder_comm, IERROR)
```

Im folgenden Beispiel ist eine nicht effiziente Strategie gezeigt (genaueres siehe Kapitel Lastausgleich).

```
!--- MPI
      !communicator for MPI
      INTEGER :: old_comm, cylinder_comm, ndims, reorder, IERROR, nprocs,  &
                                                 my_rank, faktor, long_end
      INTEGER dim_size(2)
      INTEGER :: help_dim = 0                                              !Es liegt ein
      LOGICAL periods (0:1)                                                zweidimensionales Gitter vor
      ndims = 2

      CALL MPI_Comm_Size(MPI_COMM_WORLD, nprocs, IERROR)

      old_comm = MPI_COMM_WORLD

      dim_size(1) =   1                       !longs = 360                 !Sehr schlechte Aufteilung nur
      dim_size(2) =   nprocs                  !lats = 90                   in Scheiben, nur falls 90
                                                                           ganzzahlig durch 90 teilbar
                                                                           und nur wenn nprocs < 91
                                                                           (siehe mehr in Lastausgleich)
      periods(0)  = .true.        !row-periodic                            !Ringkommunikation horizontal,
      periods(1)  = .false.       !column-nonperiodic                      aber nicht vertikal
      reorder     = 1

      CALL MPI_Cart_create(old_comm, ndims, dim_size, periods,             &  ! Neue Topologie erzeugen
                           reorder, cylinder_comm, IERROR)
      CALL MPI_COMM_SIZE(cylinder_comm, nprocs, IERROR)                       !nprocs im neuen Kommunikator

      CALL MPI_COMM_RANK(cylinder_comm, my_rank, IERROR)                      !Ränge im neuen Kommunikator
```

Zum Kommunizieren innerhalb der Topologie braucht man nun nicht mehr explizit den Rang des Nachbarn bzw. des Zielprozesses zu ermitteln. Es kann sich entsprechend der Anwendung an den Aufbau orientiert werden. Mit der Richtungsangabe spricht man die benötigten Prozesse an. Folgende Funktion ermittelt ausgehend von dem angegebenen *rank* den gesuchten:[12]

MPI_Cart_shift

Returns the shifted source and destination ranks, given a shift direction and amount
Synopsis

```
MPI_CART_SHIFT(COMM_CART, DIRECTION, DISPL, RANK_SOURCE, RANK_DEST, IERROR)
               INTEGER COMM_CART, DIRECTION
               INTEGER DISP, RANK_SOURCE
               INTEGER RANK_DEST, IERROR
```

Input Parameters

comm
communicator with cartesian structure (handle)
direction
coordinate dimension of shift (integer)
displ
displacement (> 0: upwards shift, < 0: downwards shift) (integer)
invisible input argument: my_rank in cart

Output Parameters

12 http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Cart_shift.html

```
source
    rank of source process (integer)
dest
    rank of destination process (integer)
```

Für die gegeben Anwendung würde man im Ring z.B. wie folgt kommunizieren (Round Robin ohne offset):

CALL MPI_Cart_shift(cylinder_comm, 0, 1, src_rank, dst_rank, ierror)

Vertikal kommunizieren wir nicht periodisch:

CALL MPI_Cart_shift(cylinder_comm, 1, 1, src_rank, dst_rank, ierror)

Wenn kein Nachbar vorhanden ist, wird MPI_PROC_NULL zurückgegeben. Das kann sowohl als *source*, als auch als *destination* verwendet werden. Ein Aufruf mit nur einem Prozess würde somit nicht abstürzen.

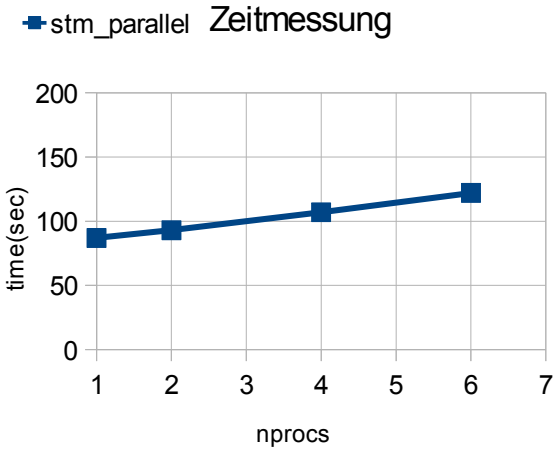
6.2 Nachrichtenaustausch

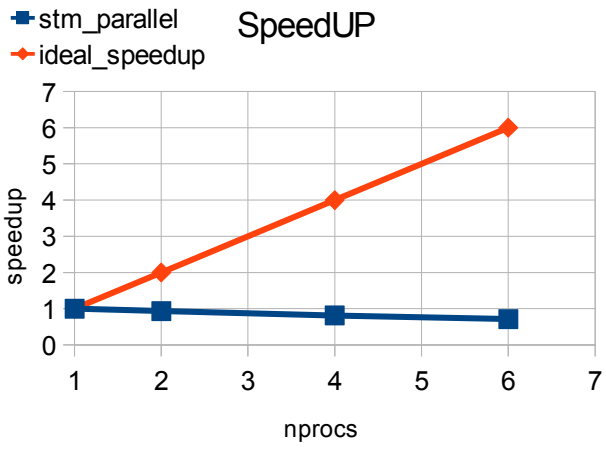
Die Idee bei dieser Anwendung ist gewesen, in der gegebenen Schleife die Daten auf die Prozessoren aufzuteilen. Man beginne hierfür in der Zeile 10. In der Topologie wurde der Arbeitsbereich schon auf Prozesse aufgeteilt, entsprechend müssen die Grenzen der Schleifen angepasst werden, da jeder Prozess auf seinem eigenen Bereich rechnen soll. Allerdings merkt man schnell, dass es damit nicht getan ist. Schon bei der Initialisierung verfügt jeder Prozess über die gesamten Ausgangsdaten. Eine gute Parallelisierung impliziert optimale Speichernutzung. Im Idealfall sollte kein Prozess das globale Array haben.
In Zeile 10 wird **`residuum`** berechnet. Es müsste umdeklariert und die Berechnungen überall umgeschrieben werden. Das Problem pflanzt sich fort und löst eine Kettenreaktion aus. Schreibt man eines der Arrays um, müssen alle anderen Berechnungen angepasst werden. Werden einmal die Grenzen der Schleifen angepasst, müssen alle darin verwendeten Arrays umdeklariert werden. Im oben angefügten Diagramm ist zu erkennen, dass die Struktur des Programms nicht erlaubt ein Modul gekapselt zu parallelisieren. Während des Umschreibens ist es sehr leicht viele Fehler einzubauen, die ab einem gewissen Moment kaum zu überblicken sind. Da dies im gegebenen Rahmen nicht gelungen ist, kann die Speichereffizienz an dieser Stelle leider nicht berücksichtigt werden.
Die Schleife wird dennoch an die lokalen Prozessgrenzen angepasst. Obwohl jeder Prozess die globalen Matrizen besitzt, soll er nur die lokalen Bereiche überschreiben. Um anschließende Rechnungen nicht zu verfälschen, müssen die berechneten Daten entweder unter allen Prozessen ausgetauscht werden, so dass jeder Prozess die globalen und aktuellen Daten besitzt und weiteren Code unverfälscht ausführen kann. Eine **`Alltoall`** Kommunikation ist sehr teuer, da hier vor allem auch verhältnismäßig große Datenmengen ausgetauscht würden. Alternativ könnte versucht werden, den restlichen, nicht parallelisierten Code nur vom 0. Prozess ausführen zu lassen, der die lokal berechneten Daten z.B. mittels **`Gather`** einsammelt. Es ist jedoch kaum umsetzbar 98% des Codes in einem Block zu kapseln, der nur von einem Prozess ausgeführt werden soll, während alle anderen warten.
In dem gegeben Rahmen gibt es keine optimale Lösung, die Gewinn versprechen könnte. Die Teilschritte sind für sich genommen richtig, jedoch können sie nicht gekapselt werden.
Um einen in sich abgeschlossenen Schritt aufzustellen, wird eine Schleife parallelisiert und am Ende auf Kosten der Zeit und des Speichers an alle Prozesse versandt. Hierbei bleibt zumindest die Korrektheit bewahrt.
Bei der Parallelisierung der Schleife werden die Grenzgebiete nach jeder Iteration versandt. Um die Grenzgebiete zu bestimmen, muss die Datenabhängigkeiten berücksichtigt werden. Sofern die Aufteilung in der Topologie nur auf 90 Prozesse beschränkt ist und die *longitudes* nicht betrifft, sind nur Breiten relevant.
An dieser Stelle kam es während der Arbeit zum fatalen Verwechseln der Längen und Breiten im Code. Daher wurde nicht nur das Ergebnis falsch berechnet, sondern ein Teil der Daten gar nicht berechnet. Zum Korrigieren soll entgegen des Vorhabens auf die *longs* zugegriffen werden, jedoch angepasst an die tatsächliche Größe von 360°. Die Fehler beim Parallelisieren der *lats* waren wegen komplexer Anpassung an die Erdhalbkugeln in dem Rahmen nicht mehr zu beheben.

```
1   lon = longs_start + (rank * 360 / nprocs)       !Beginn der Längen abhängig vom Rang
2   mpi_end = lon + (360 / nprocs)                  !Ende der Längen abhängig vom Rang
3
4   !--- start iteration
5   DO count_checks = 0 , max_convergence_checks
6     DO count_iteration = 1 , number_iterations
7       !--- set southerly hemisphere to iterate first
8       hemisphere = 1
9       start_lat  = start_south_lat + 1
10      end_lat    = end_south_lat
11      DO WHILE ( hemisphere < 3 )
12        !--- latitudes without boundaries
13        DO lat = start_lat, end_lat
14           DO lon = longs_start, mpi_end
15              residuum         (lon , lat)                          = &
16              coefficient_north(lon , lat) * zeta_new(lon   , lat+1) + &
17              coefficient_south(lon , lat) * zeta_new(lon   , lat-1) + &
18              coefficient_east (lon , lat) * zeta_new(lon+1 , lat  ) + &
19              coefficient_west (lon , lat) * zeta_new(lon-1 , lat  ) - &
20              zeta_new(lon   , lat  ) + &
21              coefficient_zeta (lon , lat)
22              count = count +1
23           END DO
24       END DO
25
26       CALL MPI_Alltoall(residuum(lon,start_lat), count, MPI_REAL, residuum,          &
27                         count * (nprocs-1), MPI_REAL, cylinder_comm, IERROR)
28
29        !--- set northerly hemisphere to iterate next
30        hemisphere = hemisphere + 1
31        start_lat  = start_north_lat
32        end_lat    = end_north_lat - 1
33      END DO
34
35  …
36
```

6.3 Leistungsanalyse

Aus oben genannten Gründen ist von vorne rein keine Verbesserung der sequentiellen Zeit zu erhoffen, was sich bei den Zeitmessungen auch bestätigt. Die Zeit steigt nahezu geradlinig nach oben. Das hängt sicherlich mit dem Kommunikationsaufwand zusammen. Je mehr Prozesse an der Berechnung beteiligt sind, umso größer wirkt sich die **`Alltoall`** Kommunikation aus. Es lohnt sich nicht die Berechnung auf Kosten der Kommunikation aufzuteilen. Bei dem klaren Verlauf wurden entsprechend keine höhere Anzahl der Prozesse getestet.





7. Lastausgleich

Die Lastverteilung kann in diesem Modell unter zwei Aspekten berücksichtigt werden. Zum Einen geht es um die Verteilung der Wasser- und Landpunkte bei der Topographie. Die Landpunkte sollen bei der Berechnung nicht berücksichtigt werden. Beim Verteilen der Daten auf die Prozessoren sollte möglichst darauf geachtet werden, dass jeder Prozess gleich viel zu berechnen hat. Eine optimale Verteilung der Wasserpunkte könnte aber mit sich führen, dass die Grenzgebiete der jeweiligen Prozesse so ungünstig sind, dass die Zahl der Kommunikationspartner ansteigt. Dabei kommt es auf die Datenabhängigkeit an. In dieser Anwendung werden die Daten aus 4 Richtungen benötigt. Folgendes Beispiel zeigt schematisch die Balance zwischen guter Lastverteilung und angemessenem Kommunikationsaufwand:

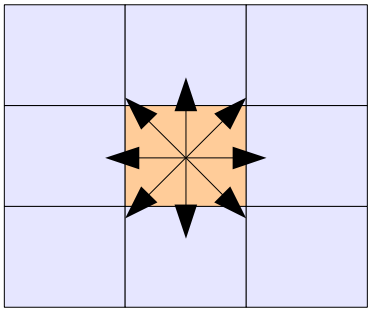


Abb. 6.a

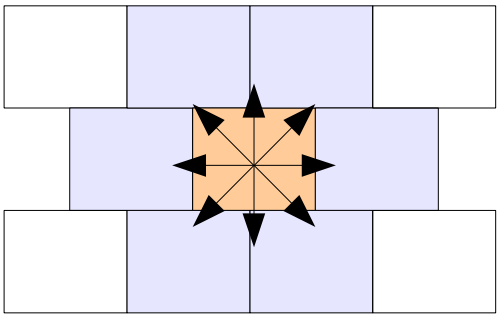


Abb. 6.b

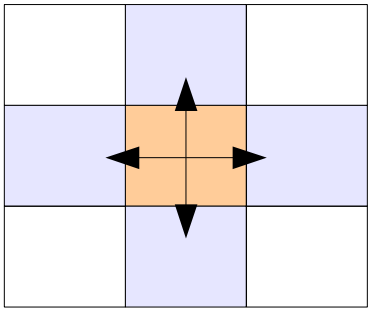


Abb 6.c

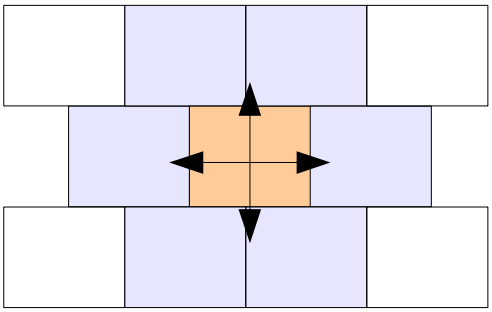


Abb. 6.d

Abbildungen 6.a und 6.b zeigen eine Datenabhängigkeit in 8 Richtungen. Das Kommunikationsschema der Abbildungen unterschiedet sich um das *„align“* - die gerade Ausrichtung zu einem kartesischen Gitter. Das Schema auf der Abb. 6.a zeigt die Notwendigkeit mit 8 Nachbarn zu kommunizieren, während die nicht gerade Topologie mit 6 Nachbarkommunikationen auskommt. Liegen jedoch nur vier Kommunikationsrichtungen vor, fordert das Kommunikationsschema auf Abb. 6.d zwei Kommunikationsnachbarn mehr, als auf Abb. 6.c.

Der zweite Aspekt der Lastverteilung hängt mit der virtuellen Topologie zusammen. Um eine optimale Verteilung vorzunehmen, muss auch hier die Datenabhängigkeit berücksichtigt werden. Das Gitter weist ein Seitenverhältnis von 4:1 auf, wobei die maximale Breite bei 360 und Höhe bei 90 liegen, da pro Grad und Gitterzelle je eine Berechnung anfällt. Es kann theoretisch maximal eine Berechnung pro Prozess gemacht werden, wobei dies schon unangemessenen Kommunikationsaufwand bewirken würde.
Eine Strategie könnte dabei sein, die Daten entsprechend dem Seitenverhältnis aufteilen (siehe Abb. 7). Dabei treten jedoch entscheidende Nachteile in den Vordergrund. Einerseits muss die Anzahl der benutzbaren Prozessoren dem Quadrat gerader Zahlen entsprechen. Die angegebene Zahl wird dabei auf die nächst geringere Stufe abgerundet und der Rest bleibt unbenutzt. Zum anderen widerspricht das entstehende Kommunikationschema dem ursprünglichen Gedanken, den Kommunikationsvorgang zu vereinfachen.

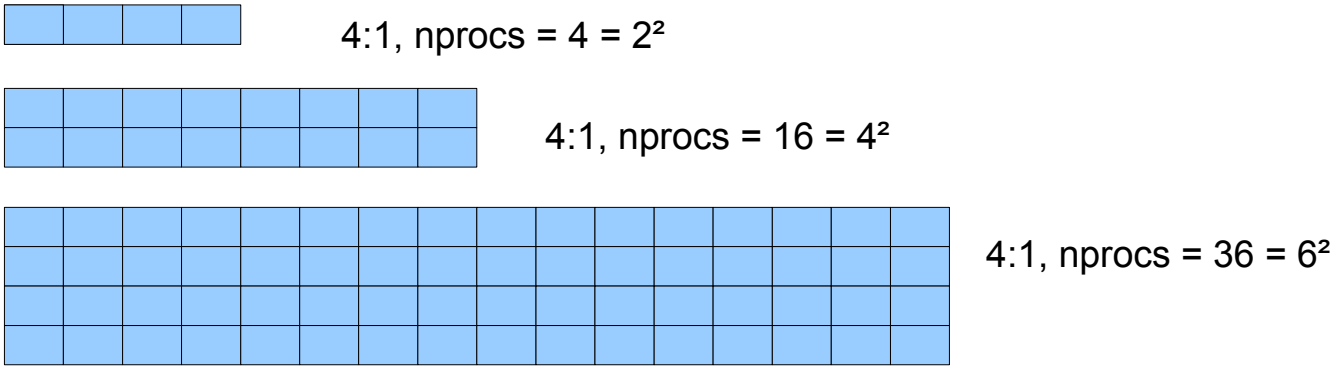


Abb. 7 Aufteilung entsprechend dem Seitenverhältnis

Eine andere Strategie ist effizienter. Man teilt die Daten im ersten Schritt scheibenweise auf. Es kann anwendungsbedingt maximal 90 Scheiben geben (das Modell lässt sich anders konfigurieren, das Gitter lässt sich beliebig verfeinern). Die Aufteilung kann nach dem Prinzip *Round Robin* vorgenommen werden. Sollte die Division 90 / *nporcs*[13] nicht ohne Rest vorgenommen werden können, so wird der Rest in einer weiteren Runde möglichst gleich auf die Prozesse aufgeteilt. Für den Fall, dass mehr als 90 Prozessoren verfügbar sind, werden nun die Scheiben möglichst gleich aufgeteilt (Abb. 8.a). Dabei ist es optimierungsabhängig, ob man erst in ganze Scheiben aufteilt, und diese später verfeinert, oder ob man sofort in Rechtecke teilt (Abb. 8.b). Wenn man die einzelnen Ringe komplett auf einen Knoten mit lokalem Speicher unterbringen kann, so kann die Kommunikation im Ring wesentlich günstiger sein und sofortige Aufteilung der Scheiben effizienter machen, falls nach „oben/unten“ weniger Kommunikation erforderlich ist.

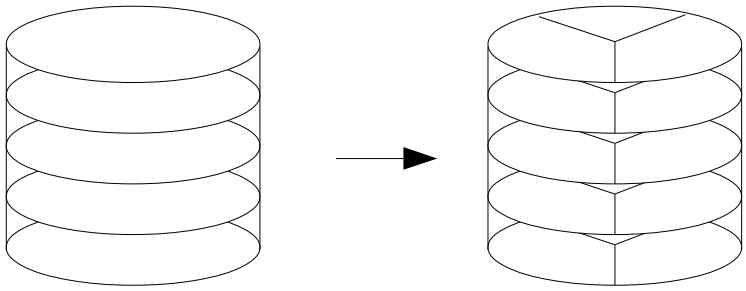


Abb. 8.a

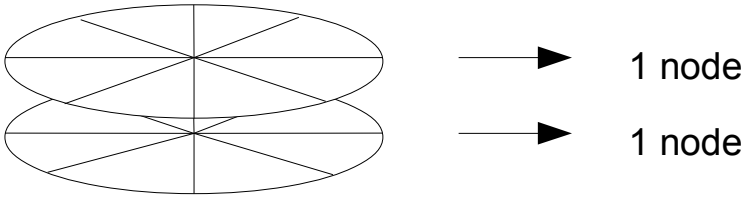


Abb. 8.b

Um die Daten in ähnlich große, möglichst quadratische rechteckige Blöcke zu unterteilen muss man die Anzahl der Prozesse faktorisieren.

nprocs	Seitenverhältnis x:y
1	1:1
2	1:2, 2:1
3	1:3, 3:1
4	1:4, 2:2, 4:1
5	1:5, 5:1
10	1:10, 2:5, 5:2, 10:1
15	1:15, 3:5, 5:3, 15:1

13 Anzahl der Prozessoren

16	1:16, 2:8, 4:4, 8:2, 16:1
20	1:20, 2:10, 4:5, 5:4, 10:2, 20:1

Wählt man *nprocs* = Primzahl, gibt es nur 1 (und gespiegelte) Variante der Aufteilung. Gibt es mehrere Möglichkeiten, so muss man sich für eine entscheiden. Bei gewünschter quadratischer Aufteilung sollte das Verhältnis der Seiten 1 anstreben. Es wäre jedoch ineffizient, vor allem bei größeren Anzahl der Prozesse, zu faktorisieren und bei jedem Paar das Verhältnis zu überprüfen. Eine bessere Umsetzung ist, erst die Wurzel aus *nprocs* zu ziehen (nach unten abgerundet), und ausgehend von dieser sich zur nächst möglichen ganzzahligen Aufteilung vorzuarbeiten (Beipiel in C):

```
1   …C !!!
2
3   int nprocs;
4   MPI_Comm_size (MPI_Comm_cy, &nprocs);          // Anzahl der Prozesse im Topologie Kommunikator
5   int snprocs = sqrt(nprocs);                    // rundet implizit nach unten ab
6   int x;                                         // Seitenverhältnis x:y
7   int y;
8   for (int i = snprocs; i>0;  i--){
9      if (nprocs % i == 0){
10        x = i;
11        y = nprocs / i                          //Sobald nprocs durch die von snprocs nächst
12        break();                                //mögliche Ganzzahl teilbar ist, ist dies
13     }                                          //das dem Quadrat nächstmögliche
14  }                                             //Seitenverhältnis
15
16  …
```

Manchmal ist es jedoch gewünscht, das Seitenverhältnis nahe dem Seitenverhältnis der Daten zu halten. Hierbei ist eine Erweiterung der o.g. Strategie nötig. Man gehe wieder von dem Wurzelbetrag aus. Hierbei sollte man jedoch nicht das erste passende Seitenverhältnis nehmen, sondern sich dem entsprechenden Optimum nähern (Beispiel in C):

```
1   … C !!!
2
3   int nprocs;                                                 Anzahl der Prozesse in der Topologie
4   MPI_Comm_size (MPI_Comm_cy, &nprocs);                       Kommunikator
5   int snprocs = sqrt(nprocs);                                 rundet implizit nach unten ab
6   int x_leng, y_leng;         //Anzahl der Blöcke nach Aufteilung
7   int h = 180;                //globale Höhe
8   int w = 360;                //globale Breite                //Sobald nprocs durch die von snprocs
9                                                               nächst //mögliche Ganzzahl teilbar ist,
10  // faktorisieren nprocs                                     ist dies
11                                                              //das dem Quadrat nächstmögliche
12  double prop = 1;            //globales Seitenverhältnis     //Seitenverhältnis
13  int tmp_x = -1;             //temp Blockzahl in x Richtung
14  double tmp_prop = -1.0;     //temp Seitenverhältnis zum Vergleich
15
16  prop = (h < w) ? (double)h/(double)w : (double)w/(double)h;  Seitenverhältnis bilden immer <= 1
17  for (int f = snprocs; f > 0; f-- ){                         Von Wurzel ausgehend runterzählen
18    if (nprocs % f == 0){                                     Wenn nprocs ganzzahlig teilbar ist
19      x_leng = nprocs / f;
20      y_leng = f;
21                                                              Betrag der Differenz (globales
22      double actu = fabs (prop - (double)y_leng/(double)x_leng);  Seitenverhältnis und aktuelle Aufteilung)
23
24      double past = fabs (prop - tmp_prop) ;                  Betrag der Differenz (globales
25                                                              Seitenverhältnis und letzte Aufteilung)
26      if (actu < past ){
27         tmp_prop = (double)y_leng / (double)x_leng;          Wenn aktuelle Aufteilung besser als
28         tmp_x = x_leng;                                      letzte, übernimm die aktuellen Werte
29      }
30      else {
31         x_leng = tmp_x;                                      Wenn die aktuelle Aufteilung schlechter
32         y_leng = nprocs / tmp_x;                             als letzte, übernimm die letzten Werte
33         break;                                               und brich ab, besser wird's nicht mehr.
34      }
35    }
36  }
37
38  if(h > w) {
39    x_leng = y_leng;                                          Anpassung der Seitenverhältnis abhängig
40    y_leng = tmp_x;                                           davon ob Höhe oder Breite größer war.
41  }
42  …
43
```

Dabei gilt folgender Zusammenhang:
bei quadratischer Blockaufteilung erreicht man Blockgrößen im Verhältnis ähnlich dem der globalen Daten; bei Blockaufteilung entsprechend dem Verhältnis der globalen Daten erreicht man möglichst quadratische Blockgröße.
Letzteres ist öfter gewünscht.

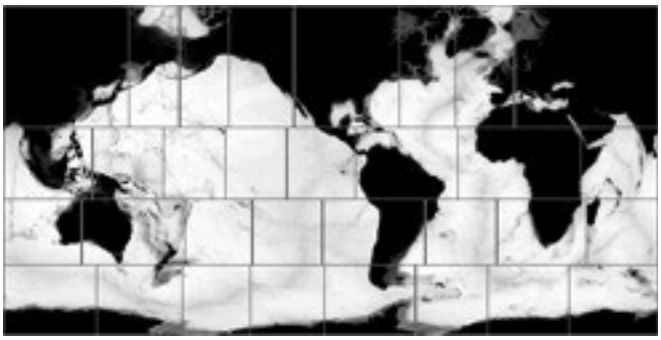


Abb. 9 Beispiel einer lastangepassten Domänendekomposition für das Ozean Modell MPIOM mit einem hierarchischen Ansatz.

So kann eine gleichmäßige Aufteilung aussehen, erstellt mit ScalES[14].

14 http://www.mpg.de/4776190/Skalierbarkeit_Erdsystemmodelle?c=5732343

8. Zusammenfassung

Die Anwendung bietet einen sehr interessanten Hintergrund. Man lernt einerseits kennen, zu welchem Zwecke die Informatik ihre Arbeit leistet, aber auch viel über die Zusammenarbeit und Abhängigkeit der wissenschaftlichen Gebiete.

Zu Beginn der Arbeit wird ein Einstieg in die Anwendung und das Modell gewährt. Dabei wird der für die Parallelisierung nötige mathematische und physikalische Hintergrund erläutert. Am Moduldiagramm wurde die Abhängigkeitsstruktur des Modells gezeigt. Simulation der Gezeiten ist ein umfangreiches und facettenreiches Problem. Das Differentialgleichungssystem kann hier nur numerisch gelöst werden. Das vorgelegte Modell ist eine der Möglichkeiten dies zu simulieren.
Die Datenabhängigkeitsstruktur lässt Platz für viele Umsetzungsideen. Für die Parallelisierung wurde eine Zylindertopologie aufgestellt, die intuitiveres Kommunizieren bewirkt. Es ist zwar nicht notwendig, eine aufzustellen, zudem das Konzept keinen guten Ruf hinsichtlich der Effizienz genießt, in der Komplexität des Problems war jedoch Klarheit und Transparenz priorisiert. Die Topologie ist ein Abbild einer Halbkugel. Um volle Prozessbenutzung zu erreichen wird gleichzeitig auf einer Topologie, aber abwechseln auf den Daten der Süd- und Nordhalbkugel gerechnet. Durch eine unglückliche Verwechslung musste leider von der Grundidee abgewichen werden. Es wurde nach den geographischen Längen aufgeteilt, statt wie erwünscht und in der Topologie vorbereitet nach Breiten. Die komplexeren Zusammenhänge der Berechnungen in verschiedenen Module hatten zur Konsequenz, dass jeder Prozess die gesamten Daten hält und darüber hinaus nach jeder Iteration seinen gesamten lokalen berechneten Bereich an alle Prozesse in der Topologie verschickt. Diese teuren Operation begründen die schlechte Laufzeit. In der Analyse sieht man deutlich, dass sich der Kommunikationsaufwand bedeutender auswirkt, als der Gewinn durch Datenaufteilung. Die Zeiten steigen mit jedem weiteren beteiligen Prozess. Über das Grundproblem der engen Kopplung der Daten im Programm hinaus, lässt sich der Lastausgleich verbessern. Es wurden Überlegungen über die optimale Verteilung der Wasser- und Landpunkte angestellt, sowie über verschiedene Aufteilungsstrategien innerhalb der Topologie. Beispielhaft sind zwei Strategien in C aufgeführt, die das Seitenverhältnis der globalen Daten berücksichtigen und dabei möglichst quadratische Teilblöcke generieren.

Während der Arbeit sind Probleme aufgetreten, auf Grund derer keine abgeschlossene Parallelisierung möglich war. Das gewählte Konzept erlaubt leider keine Kapselung der in sich korrekten Schritte, so dass kontraproduktive Maßnahmen, wie die **`Alltoall`** Kommunikation notwendig waren.
Die Arbeit lehrt die Grenzen besser abzuschätzen und die Gesamtheit des Problems im Auge zu behalten. Darüber hinaus ist die Wichtigkeit der richtigen Balance zwischen der Kommunikation und dem Rechnen noch einmal deutlich geworden.

9. Anhang A: Arbeitsmittel

Verwendete Hardware:
Cluster[15] *aus 10 Knoten, jeder Knoten besitzt folgende Ausstattung:*
2 Prozessoren (Intel Xeon Westmere 5650 @ 2.67GHz) mit jeweils 6 cores
12 GByte DDR3 / PC1333 Hauptspeicher (System taktet mit 1333 MHz)
2 x Gigabit-Ethernet

Bei der Parallelisierung wurde die *mpich2/1.4*[16] Bibliothek und der
GNU Fortran (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3 Compiler benutzt.

Zur besseren Zusammenarbeit und Versionsverwaltung wurde die Software *git*[17] verwendet, gekoppelt an das Project-Management Tool *redmine*[18] auf dem *wr-Cluster*.

Mit *gprof*[19] wurde die Leistungsanalyse durchgeführt, die die zeitaufwendigsten Programmteile ermittelt hat.

Compilereinstellungen:

Für das Debuggen sind folgende Optionen eingestellt:
FCFLAGS = -g -O0 -std=f2003 -Wall -pedantic -fbounds-check -ffpe-trap=invalid,zero,overflow,underflow
LDFLAGS = -g -O0 -std=f2003 -Wall -pedantic -fbounds-check -ffpe-trap=invalid,zero,overflow,underflow
Für Zeitmessungen werden Debuggingoptionen abgeschaltet und -O3 eingeschaltet.
Zum Debuggen wurden *gdb*[20], *ddt*[21] und *valgrind*[22] verwendet.

Werkzeuge:

OpenOffice
doxygen

15 http://wr.informatik.uni-hamburg.de/teaching/ressourcen/cluster
16 http://www.mcs.anl.gov/research/projects/mpich2/
17 http://git-scm.com/
18 http://www.redmine.org/
19 http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html
20 http://www.gnu.org/software/gdb/
21 http://www.allinea.com/products/ddt/
22 http://valgrind.org/

10. Quellen

Literaturverzeichnis

Nerge, P. (1998). Resonanzeigenschaften der globalen Ozeangezeiten für Topographien der Gegenwart und des Proterozoikums – Gezeitendrehmoment und die Entwicklung des Systems Erde-Mond während der Erdgeschichte. Diplomarbeit, Institut für Meereskunde, Universität Hamburg. 117p.

Seiler, U. (1989). An Investigation to the Tides of the World Ocean an their Instantaneous Angular Momentum Budgets. Mitteilungen des Institutes für Meereskunde der Universität Hamburg Nr. 29.

Wenzel, H. G. (1995). Tidal Rhytmites: Geochronometers for the Ancient Earth-Moon System. *Episodes*, 12(3), 162-171.

Williams, G. E. (2000). Geological Contraints on the precambrian History of Earth's Rotation and Moon's Orbit. Reviews of Geophysics, 41(3), 391-411.