Offshore-Windparks: Parallelisierung von Analysewerkzeugen von Ökosystem-Langzeitläufen in Fortran

— Projektbericht —

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Felix Hoffmann
E-Mail-Adresse:	2hoffman@informatik.uni-hamburg.de
Matrikelnummer:	6429010
Studiengang:	B.Sc. Wirtschaftsinformatik
Betreuer:	Dr. Hermann Lenhart

Hamburg, den 17.05.2015

Inhaltsverzeichnis

1. Ausgangssituation

In sogenannten Totzonen führt Sauerstoffmangel (Hypoxie) zum Tod von Fischen und bodenlebenden Wirbellosen wie etwa Würmern. Dem Projekt geht die Annahme voraus, dass Offshore-Windparks zum Abfall der Sauerstoffkonzentration beitragen konnten. Diese Annahme basiert auf der Abfolge ökologischer Prozesse, die zu einem Sauerstoffdefizit führen und in den Jahren um 1980 im Rahmen der Eutrophierungsproblematik in der Nordsee auftraten.[1] Zur Verarbeitung der Simulation mit ECOHAM4 entstand im Rahmen des Projekts ein Analysewerkzeug, das es erlaubt, Langzeitläufe über viele Jahre effektiv auf Hypoxie zu untersuchen. Weiterhin galt es, den O_2-Gehalt in einem Bereich der zentralen Nordsee über einen Zeitraum von 37 Jahren (1970-2006) zu untersuchen. Die benötigte Datengrundlage liefert das marine Ökosystem-Modell ECOHAM4,[2] das unter anderem den Sauerstoffgehalt im Wasser simuliert. Es stehen Daten für 24 Modellschichten von 5 bis 3500 Metern Tiefe zur Verfügung. Die Daten liegen jahrweise im Network Common Data Format (NetCDF) vor, einem Dateiformat für den Austausch wissenschaftlicher Daten.

Ein in der Sprache Fortran geschriebenes paralleles Programm nimmt diese Analyse vor. Für jeden im Modell dargestellten Punkt der Nordsee zählt es die Anzahl der Tage, an denen der Sauerstoffgehalt am Meeresboden unter einem Schwellenwert von sechs Milligramm pro Liter lag. Als Ergebnis erstellt das Programm für jedes Jahr eine zweidimensionale Matrix, die für jeden Punkt die Summe der betroffenen Tage speichert.

Für bessere Lesbarkeit fasst es abschließend die Ergebnisse zu aussagekräftigen Durchschnittswerten zusammen, die den Vergleich zwischen den Jahren erlauben.

[1] [Gre10] Vgl. N. Greenwood et al. Detection of low bottom water oxygen concentrations in the North Sea; implications for monitoring and assessment of ecosystem health, 2010.

[2] [Lor12] Vgl. I. Lorkowski et al. Interannual variability of carbon fluxes in the North Sea from 1970 to 2006 – Competing effects of abiotic and biotic drivers on the gas-exchange of CO2, 2012.

2. Hilfsprogramm BottomOfSea

Für die Auswertung bedarf es für jeden Punkt der Information, in welcher Schicht sich der Meeresboden befindet. Die darüberliegende Schicht ist von Interesse. Um die entsprechende Schicht für die einzelnen Punkte zu ermitteln, kommt ein Hilfsporgramm zum Einsatz. Das *BottomOfSea* betitelte Programm bedient sich am Modul *mod_netcdf_read*, das das Einlesen von NetCDF-Dateien ermöglicht. Die aus der Datei gewonnene Datengrundlage wird in einem dreidimensionalen Array gespeichert, das die Parameter Breitengrad, Längengrad und Schicht enthält. Jeder Eintrag enthält den Sauerstoffgehalt in Millimol pro Kubikmeter für den entsprechenden Punkt und die entsprechende Schicht. Einträge, die Land oder Meeresboden symbolisieren, sind mit dem Wert -9999 belegt. Mithilfe dieser Einträge wird die letzte Meeresschicht identifiziert.

Listing 2.1: Datenstruktur

```
1 real, dimension(jmax, imax, kmax) :: var
2 integer :: latitude, longitude, day=1, ierr, layer = 1
3 integer, dimension(jmax, imax) :: array
```

Für die Iteration durch das dreidimensionale Array kommen die Integer-Variablen *latitude*, *longitude*, *day* und *layer* zum Einsatz. Die Variable *ierr* wird vom Modul *mod_netcdf_read* verwendet um mögliche Fehler-Codes auszugeben. Als Ergebnis produziert das Programm ein zweidimensionales Array, das für jeden Breiten- und Längengrad speichert, in welcher Ebene sich die letzte Meeresschicht befindet. Diese Matrix ist mit *array* betitelt.

In dem Programm kommt der Datensatz aus dem Jahr 1984 zum Einsatz – dieser dient als Stellvertreter für alle Jahrgänge, da sich die Beschaffenheit des Meeresboden über den betrachteten Zeitraum nicht oder nur unwesentlich verändert. Die gewonnenen Erkenntnisse über die letzte Meeresschicht haben somit allgemeine Gültigkeit.

Der Kern des Hilfsprogramms besteht aus drei verschachtelten Schleifen. Sie iterieren durch Breitengrade, Längengrade und Meeresebenen. Das Programm sucht nach Einträgen, die einen Wert kleiner als -9000 aufweisen, da Land- und Bodenpunkte mit dem Wert -9999 belegt sind. Findet es einen solchen Eintrag in der ersten Ebene, handelt es sich um einen Landpunkt. Es speichert an dieser Stelle eine Null in der Ergebnismatrix. Taucht ein solcher Eintrag in einer anderen Ebene auf, ist die darüberliegende Ebene von Interesse und wird in der Ergebnismatrix gespeichert. Die Idee dahinter ist simpel: Geht man an einem beliebigen Punkt des Meeres in die Tiefe bis man auf Boden stößt, ist nicht der Boden sondern die darüberliegende Meeresschicht von Bedeutung.

Ein dritter Fall tritt auf, wenn auch die letzte im Modell dargestellte Ebene eine Meeresschicht ist. In diesem Fall lässt sich anhand des Modells nicht sagen, wie tief die

Nordsee an der bestimmten Stelle ist. Der Einfachheit halber wird die 24. Ebene als letzte Meeresschicht betrachtet.

Listing 2.2: Iteration durch das Array

```
1  longiloop: do longitude=1,jmax
2      latiloop: do latitude=1,imax
3          layerloop: do layer=1,kmax
4              if(var(longitude, latitude, layer) < -9000
                   ↪ .AND. layer == 1) then
5                  array(longitude,latitude) = 0
6                  exit layerloop
7              else if(var(longitude, latitude, layer) <
                   ↪ -9000) then
8                  array(longitude,latitude) = layer-1
9                  exit layerloop
10             else if (var(longitude, latitude, layer) > 0
                   ↪ .AND. layer == 24) then
11                 array(longitude,latitude) = 24
12                 exit layerloop
13             end if
14         end do layerloop
15     end do latiloop
16 end do longiloop
```

Die gewonnenen Informationen über die Beschaffenheit des Meeresboden werden in einem zweidimensionalen Array gespeichert, aufgeteilt nach Breiten- und Längengraden. Damit die Werte nicht fortlaufend, sondern formattreu in einer Matrix gespeichert werden, kommt neben der Schleife für die Breitengrade eine implizite Schleife für die Längengrade zum Einsatz. Sie sorgt dafür, dass beim Iterieren durch die Längengrade kein Zeilenumbruch zustande kommt.

Listing 2.3: Speicherung in Textdatei

```
1  do latitude=imax,1,-1
2      open(unit=15, file='RelevantLayers.txt')
3      write(15,*) (array(longitude,latitude),
           ↪ longitude=1,jmax)
4  end do
```

3. Hauptprogramm ReadData

Die Ergebnisse des Hilfsprogramms *BottomOfSea* dienen dem Hauptprogramm *ReadData* als Grundlage. Es nutzt die Informationen über die letzte Meeresschicht, da nur diese für die Analyse von Relevanz ist. Wie beim Hilfsprogramm kommt das Modul *mod_-netcdf_read* zum Einsatz. Darüber hinaus bedient sich *ReadData* beim Modul *mpi*. Die Abkürzung MPI steht für Message Passing Interface. Es ermöglicht das Senden und Empfangen von Nachrichten zwischen zwei Prozessen. Weiterhin erlaubt es, die Prozesse zu identifizieren und ihnen somit verschiedene Anweisungen im Programmcode zu erteilen. Obwohl alle Prozesse den gleichen Quelltext ausführen sorgen derartige Verzweigungen dafür, dass die Programmabläufe unterschiedlich sind.

Als Datenstrukturen kommen hauptsächlich Arrays zum Einsatz. Die letzte Meeresschicht wird im Array *input* gespeichert. Die aus der NetCDF-Datei gewonnenen Informationen werden im dreidimensionalen Array *var* gespeichert. Dort sind die Dimensionen Breitengrad, Längengrad und Meerestiefe abgebildet. Die von den Subprozessen errechneten Ergebnisse werden in *output* gespeichert. Der Haupt-Prozess vereint diese Ergebnisse in der Matrix *results*, die über die Dimensionen Jahr, Breitengrad und Längengrad verfügt. Das Schlüsselwort *allocatable* bewirkt, dass die Größe der Dimensionen erst später unter Einbezug anderer Variablen festgestelt werden kann. Das eindimensionale Array *status* beinhaltet Meta-Daten einer MPI-Nachricht. So lässt sich etwa mit *status(MPI_SOURCE)* auf den Absender eine Nachricht zugreifen.

Listing 3.1: Datenstruktur und Allokation

```
1 real, dimension(jmax, imax, kmax) :: var
2 integer, dimension(MPI_STATUS_SIZE) :: status
3 integer, dimension(jmax, imax) :: input, output
4 integer, dimension(:,:,:), allocatable :: results
5
6 min_year = 1970
7 max_year = 2006
8
9 allocate(results(max_year-min_year+1,jmax-60,imax-60))
```

Programme, die das Message Passing Interface verwenden, müssen dieses zunächst initialisieren. Anschließend lässt sich den Prozessen ein Rang zuordnen – über ihn sind sie eindeutig zu identifizieren. Die Anzahl der Prozesse wird vom Nutzer beim Programmaufruf festgelegt. Damit auch das Programm selbst die Anzahl der Prozesse kennt, fragt es sie mit dem Befehl *MPI_COMM_SIZE* ab und speichert sie in der Integer-Variable *max_rank*.

Listing 3.2: Initialisierung des Message Passing Interface

```
1 call MPI_INIT(ierr)
2 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
3 call MPI_COMM_SIZE(MPI_COMM_WORLD, max_rank, ierr)
```

Zu Beginn liest der Hauptprozess mit dem Rang 0 die von *BottomOfSea* erstellten Informationen über die letzte Meeresschicht in. Die Matrix mit den entsprechenden Ebenen – im Quelltext mit der Variable *input* bezeichnet – wird mit dem Befehl *MPI_BCAST* an alle Subprozesse gesendet. Sie benötigen die Informationen für die weitere Analyse.

Listing 3.3: Einlesen und Broadcast der Bodenschichten

```
1  if (rank .eq. 0) then
2      open (unit=15, &
3      file='/home/hoffmann/BottomOfSea/RelevantLayers.txt', &
         ↪              status='old', action='read')
4
5      do latitude=imax,1,-1
6          read(15,*) (input(longitude,latitude), longitude=1,
             ↪ jmax)
7      end do
8  endif
9
10 call MPI_BCAST(input, imax*jmax, MPI_INTEGER, 0,
     ↪ MPI_COMM_WORLD, ierr)
```

Anschließend verteilt der Hauptprozess die ersten Jahrgänge an die Subprozesse. Die initiale Zuteilung der Aufgaben erfolgt liniar: Bei x Subprozessen erhalten diese nacheinander die ersten x zu bearbeitenden Jahre. Für das Projekt galt es die Datensätze von 1970 bis 2006 zu untersuchen; dabei kamen elf Subprozesse zum Einsatz. Zu Beginn erhält Prozess 1 demnach das Jahr 1970 und Prozess 2 das Jahr 1971. Die Zuweisung der anderen Jahre erfolgt analog bis zum elften Prozess, der das Jahr 1980 erhält.

Listing 3.4: Initiale Zuweisung der Jahrgänge

```
1 if (rank .eq. 0) then
2     year = min_year
3     do rank=1, max_rank-1
4         write(year_char,'(I4)') year
5         call MPI_SEND(year_char, 4, MPI_CHAR, rank, 1,
            ↪ MPI_COMM_WORLD,ierr)
6         year = year + 1
7     enddo
```

Die weitere Verteilung der Aufgaben erfolgt dynamisch: Immer wenn ein Subprozess durch das Senden der Ergebnisse signalisiert, dass die Berechnungen abgeschlossen sind,

weist der Hauptprozess ihm ein neues Jahr zu. Wie die Berechnungen der Subprozesse im Detail aussehen, wird an späterer Stelle erläutert. Nachfolgende Quelltext-Ausschnitte schließen an obiges Beispiel an und sind daher Teil der geöffneten If-Verzweigung, die sicherstellt, dass dieser Teil des Programms nur vom Hauptprozess ausgeführt wird.

Da der Hauptprozess nicht weiß, welcher Subprozess die Berechnungen als erstes abgeschlossen hat, muss er auf den Nachrichteneingang eines beliebigen Prozesses gefasst sein. Während der Receive-Befehl von MPI die Angabe des Absenders vorsieht, hilft der Platzhalter *MPI_ANY_SOURCE* diese Angabe zu umgehen. Parallel dazu akzeptiert der Hauptprozess durch den Platzhalter *MPI_ANY_TAG* Nachrichten mit beliebigen Tags. Beim Tag handelt es sich um eine Ganzzahl anhand der sich die Nachricht identifizieren lässt. Im vorliegenden Fall entspricht der Tag der Jahreszahl, die der Subprozess berarbeitet hat. Über die Metadaten – gespeichert im Array *status* – erhält Rang 0 neben der Nachricht auch die Informationen darüber, von welchem Prozess sie stammt und um welches Jahr es sich bei den gesendeten Ergebnissen handelt. Die Abfrage dieser Meta-Daten erfolgt über *status(MPI_SOURCE)* und *status(MPI_TAG)*. Alternativ ließe sich die Abfrage auch über die Position im Array bewerkstelligen. Die Quelle ist etwa an Position 1 hinterlegt (*status(1)*), der Tag an Position 2. Deutlich lesbarer und benutzerfreundlicher ist aber die erstgenannte Variante.

Der Hauptprozess erwartet 37 Ergebnis-Matrizen, die es im dreidimensionalen Array *results* unter Einbezug des Jahrgangs speichert. 1970 erhält den Array-Platz 1, 2006 den Platz 37. Die unten gezeigte Do-Schleife läuft dementsprechend oft durch. Nach 26 Durchläufen (37 subtrahiert mit 11, bei elf Subprozessen) empfängt Rang 0 die letzten Ergebnisse, sendet aber keine neuen Jahrgänge an die Subprozesse. Stattdessen sendet er das Signalwort *done*, das den Subprozessen vermittelt, dass alle Jahrgänge verteilt wurden.

Listing 3.5: Empfangen der Ergebnisse und Zuweisung neuer Jahrgänge

```
1     do year=year,max_year+max_rank-1
2         call MPI_RECV(output, jmax*imax, MPI_INTEGER,
              ↪ MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
              ↪ status, ierr)
3
4         do latitude=imax-36,25,-1
5             do longitude=50,jmax-11
6                 results(status(MPI_TAG)-min_year+1,
                      ↪ longitude-49, latitude-24) =
                      ↪ output(longitude,latitude)
7             end do
8         end do
9
10        if (year > max_year) then
11            year_char = 'done'
12        else
13            write(year_char,'(I4)') year
```

```
14          endif
15
16          call MPI_SEND(year_char, 4, MPI_CHAR,
              ↪ status(MPI_SOURCE), 1, MPI_COMM_WORLD, ierr)
17      enddo
```

Nachdem Rang 0 alle Ergebnisse im Array *results* abgelegt hat, folgt die Zusammenfassung dieser. Für jeden Jahrgang errechnet das Programm Summe, Maximum und Ausdehnung. Für die Summe der Tage mit Sauerstoffdefizit addiert es alle Einträge, deren Wert kleiner als 9999 ist. 9999 markiert Landpunkte – diese sollen nicht mitgezählt werden. Um die Ausdehnung zu bestimmen, zählt es die Punkte, bei denen es mindestens an einem Tag zu einer Sauerstoffkonzentration von weniger als sechs Milligramm pro Liter kam. Für die Bestimmung des Maximums kommt die Funktion *maxval* zum Einsatz, die den höchsten Wert der Matrix zurückgibt. Auch Ausdehnung und Maximum schließen die Landpunkte bei der Berechnung aus.

Die zusammengefassten Ergebnisse werden im CSV-Format gespeichert. Die Trennung durch Semikolons erlaubt Tabellenverarbeitungsprogrammen wie Microsoft Excel ein einfaches Einlesen der Daten. Nach dem Einlesen können etwa Diagramme erstellt werden, die die Ergebnisse grafisch aufbereiten. Ein Diagramm mit den hier beschriebenen Ergebnissen befindet sich im nächsten Abschnitt des Projektberichts. Neben den aggregierten Ergebnissen werden auch die Matrizen mit den detaillierten Angaben zum Sauerstoffgehalt pro Punkt und Jahr in einer CSV-Datei gespeichert.

Listing 3.6: Speichern der Ergebnisse

```
1       open(unit=1,file='results.csv',status='unknown')
2       open(unit=2,file='data.csv',status='unknown')
3
4       write(1,*) "Year; Sum; Spread; Maximum"
5       do year=min_year, max_year
6           write(1,*) year, ";", &
7           sum(results(year-min_year+1,:,:), &
8           results(year-min_year+1,:,:) < 9999), ";", &
9           count(results(year-min_year+1,:,:) > 0 .AND. &
10          results(year-min_year+1,:,:) < 9999), ";", &
11          maxval(results(year-min_year+1,:,:), &
12          results(year-min_year+1,:,:) < 9999)
13
14          write(2,*) year
15          do latitude=(imax-61),1,-1
16              write(2,*)
                  ↪ (results(year-min_year+1,longitude,latitude),
                  ↪ &           ";", longitude=1,jmax-60)
17          end do
18      enddo
```

```
19 endif
```

Zur Erinnerung: Das abschließende endif schließt die If-Verzweigung, die den Programmteil kennzeichnet, der exklusiv von Rang 0 ausgeführt wird. Analog dazu gibt es einen Programmteil für die Subprozesse mit Rang 1 und größer. Diese Prozesse werden erst aktiv, wenn sie vom Hauptprozess eine Nachricht mit dem zu berabeitenden Jahrgang erhalten.

Anschließend beginnt das „Herzstück“ des Programms: die Berechnung der Tage mit Sauerstoffmangel. Sie besteht im Kern aus drei verschachtelten Schleifen, die durch die Tage des Jahres, Breitengrade und Längengrade iterieren. Für jeden Punkt und jeden Tag gleicht das Programm mit der *input*-Matrix ab, ob es sich um einen Land- oder Meerespunkt handelt. Liegt ein Meerespunkt vor, greift es in der NetCDF-Datei, die als Datengrundlage dient, am entsprechenden Punkt und in der von der *input*-Matrix angegebenen Ebene auf die Sauerstoffkonzentration zu. Liegt diese unter 187,5 Millimol pro Kubikmeter (das entspricht dem Schwellenwert von 6 Milligram pro Liter), erhöht das Programm in der *output*-Matrix den Wert an dieser Stelle um den Faktor 1. Liegt ein Landpunkt vor, markiert das Programm diesen mit dem Wert 9999. Nach Durchlauf der Schleifen enthält die *output*-Matrix für jeden Punkt die Anzahl der Tage, an denen der Sauerstoffgehalt unter 6 Milligramm pro Liter lag.

Listing 3.7: Iteration durch das Array

```
1  time: do day=1,365
2      longiloop: do longitude=1,jmax
3          latiloop: do latitude=1,imax
4              if (input(longitude, latitude) > 0) then
5                  if (var(longitude,latitude,input(longitude,
                       ↪ latitude)) < 187.5) then
6                      output(longitude,latitude) =
                           ↪ (output(longitude,latitude) + 1)
7              end if
8              else
9                  output(longitude,latitude) = 9999
10             end if
11         end do latiloop
12     end do longiloop
13 end do time
```

Abbildung 3.1 veranschaulicht, in welcher Reihenfolge der Hauptprozess mit den Subprozessen über das Message Passing Interface interagiert.

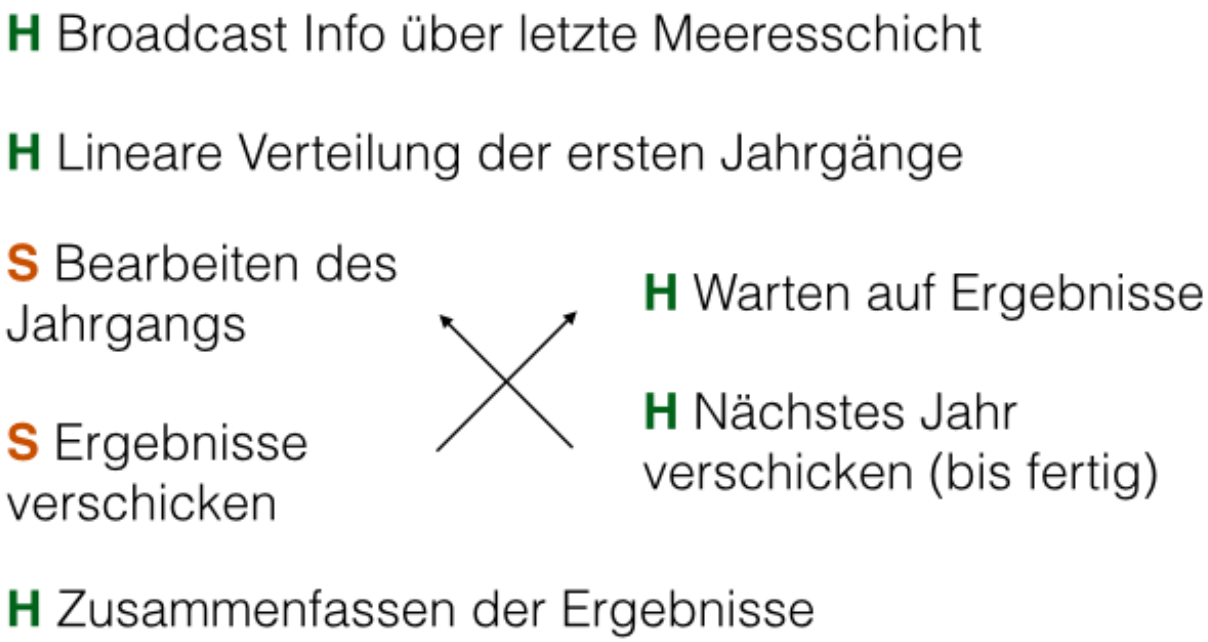

Abbildung 3.1.: Send-, Receive- und Broadcast-Befehle des Hauptprogramms. H steht für den Hauptprozess mit Rang 0, S für alle Subprozesse mit Rang größer als 0.

4. Ergebnisse

Nach der Berechnung der Tage mit Sauferstoffdefizit liegt ein dreidimensionales Array vor, in dem für die berechneten Jahre sowie die relevanten Breiten- und Längengrade der jeweilige Wert hinterlegt ist. Wird die Ausgabe auf ein bestimmtes Jahr beschränkt, symbolisiert die Matrix eine Karte der zentralen Nordsee. Landpunkte sind mit Sternchen gekennzeichnet. Bei den in Abbildung 4.1 sichtbaren Sternchen handelt es sich um Teile von Dänemark und Deutschland. Zwar liefert das ECOHAM4-Modell, das als Grundlage dient, einen weitaus größeren Ausschnitt der Nordsee. Für die geplanten Analyse-Zwecke reicht ein kleinerer Ausschnitt jedoch aus.

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0**********
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0**********
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0**********
 0  0  0  0  0  0  0  0  0  0  0  0  0  2  5  0  0  0  0  0**********
 0  0  0  0  0  0  0  0  0  0  0  0 30  7  8  0  0  0  0  0**********
 0  0  0  0  0  0  0  0  0  0  0 13 22 25 15  5 14  0  0  0**********
 0  0  0  0  0  0  0  0  0  0  0 12 24 33 27  0  4  0  0  0**********
 0  0  0  0  0  0  0  0  0  0 30 22 31 49 26 25  0  4  0  0  0********
 0  0  0  0  0  0  0 18  0 15 14 10 15 61 73  0  0  0  0  7  0********
 0  0  0  0  0  0  0 23 22 15  8  7 11 19 57  0  0 15  6  3  0********
 0  0  0  0  0 15 12 22 19 17  6  6  8 17 19 84  0 10  0  0  0********
 0  0  0  0  0  8 19 26 22 24 18 17 16 25 21 25 69  0  0 26  0  0*****
 0  0  0  0  0  0 13 26 24 28 23 20 16 26 23 30 26  0  0  0  0  0  0*
 0  0  0  0 26  0  0 28 29 31 27 15 31 25 35 28 56100  0  0  0  0*****
 0  0  0 12  0  0  6 35 33 37 25  9 33  4  2 20  8 42115  0  0  0  0*
 0  0  3  0 97  0  0  0 37 27  7  0  0  0  0  0  0  0  0 48  0  0  0*
 0  0  0  0  0 67  0  0 19 10  0  0  0  0  3  0  0  0******  0********
 0  0  0  0  0  0  0  0  0  0  0 54  0  0  0  0  0*********  0********
 0  0  0  0  0  0  0  0  0  0 34  0***********************************
 0  0  0  0  0  0  0  0  0  0  0***********************************
 0  0  0  0  0  0  0  0  0  0  0***********************************
**  0  0  0  0  0  0  0  0  0  0***********************************
**  0  0  0  0  0  0  0  0  0  0***********************************
```

Abbildung 4.1.: Ergebnis-Matrix für den Jahrgang 1986.

Da die Betrachtung von 37 Kartenausschnitten für weitere Analysen nur bedingt hilfreich ist, errechnet das Programm für jedes Jahr aggregierte Werte. Wie im vorherigen Abschnitt dargelegt, errechnet es Summe, Ausdehnung und Maximum. Grafisch aufbereitet – wie in Abbildung 4.2 – ist leicht zu sehen: Die drei aggregierten Werte erreichen ihre Höhepunkte um das Jahr 1982. Ebenfalls auffällig ist das Jahr 2002: Summe und Ausdehnung sind hier relativ hoch. Es gibt in diesem Jahr jedoch keinen

Punkt, der besonders lange einem Sauerstoffdefizit ausgesetzt ist – das Maximum ist eher durchschnittlich. Alle Werte finden sich in Tabelle A.1 im Anhang.

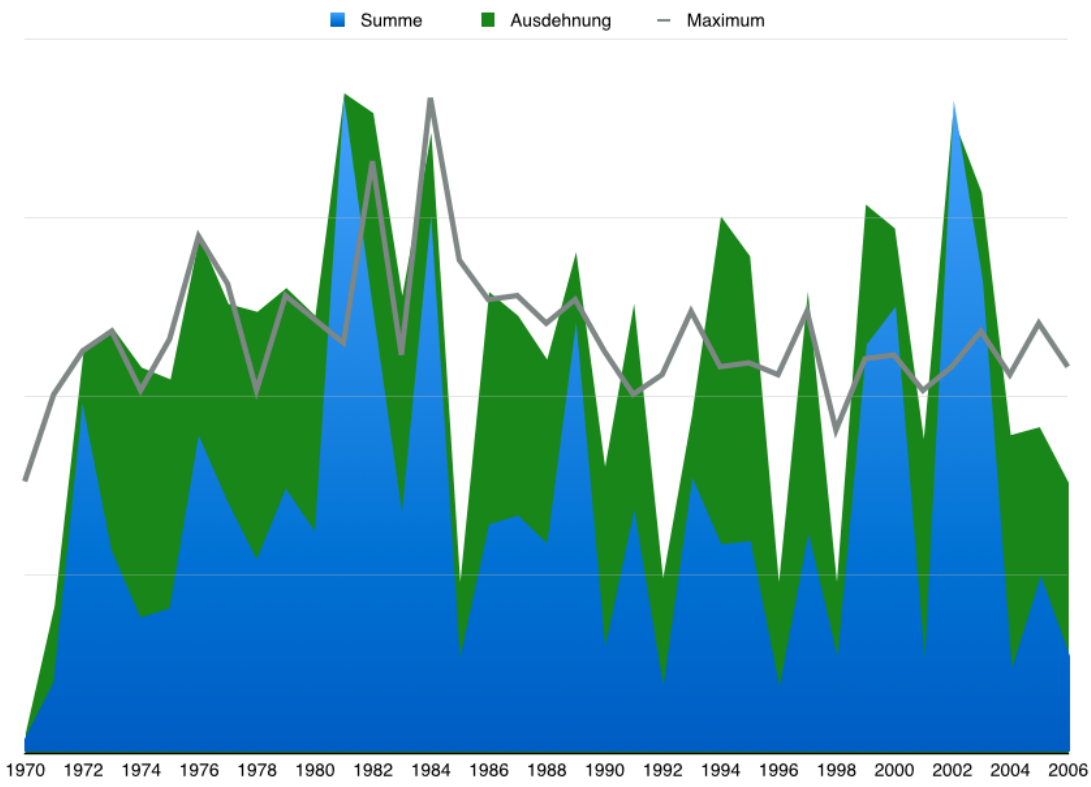

Abbildung 4.2.: Grafische Darstellung der Ergebnisse. Der jeweils größte Wert pro Kategorie entspricht 100 Prozent.

5. Probleme

Da das Projekt ohne Vorkenntnisse in Fortran, paralleler Programmierung und dem Message Passing Interface entstanden ist, sind im Laufe der Entwicklung mehrere Probleme aufgetreten. Die erste Herausforderung war es, das vom Arbeitsbereich Wissenschaftliches Rechnen bereitgestellte Beispielprogramm zu verstehen und auszuführen. Erste Anläufe scheiterten vor allem daran, dass das Konzept von Makefiles nicht bekannt war. Hinzu kam, dass die verwendete Programmierumgebung Eclipse Makefiles mit dem Befehl *make all* aufruft, der Befehl *all* in der bereitgestellten Datei jedoch nicht vorhanden war. Das Hinzufügen der Zeile *all: read_data* ergänzte das Makefile um einen Verweis auf den Hauptbefehl *read_data*, sodass Eclipse das Programm ohne Fehlermeldung kompilieren konnte.

Der nächste Fallstrick lauerte bei den ersten Gehversuchen mit dem Message Passing Interface. Zahlreiche Anleitungen beschreiben die Syntax für den Send-Befehl wie folgt:

Listing 5.1: Syntax MPI_SEND

```
1 MPI_SEND(Message, Count, Datatype, Dest, Tag, Comm, Ierror)
```

Unter der Annahme, dass der Receive-Befehl die gleichen Parameter voraussetzt und sich lediglich bei der Angabe des Absenders statt des Empfängers unterscheidet, wurde versucht erste Nachrichten zu versenden. Allerdings setzt Fortran – anders als C – einen weiteren Parameter voraus: ein Integer-Array, das Meta-Daten wie den Absender einer Nachricht sowie den verwendeten Tag beinhaltet. Jedoch gibt der Compiler keine Fehlermeldung zurück wenn der Status-Parameter ausgelassen oder statt als Array als einzelne Integer-Variable definiert wird. Statt der Nachricht wird lediglich der Tag übertragen, was zu bizarren Ergebnissen führt und die Fehlersuche erschwert. Letztendlich handelte es sich um einen Flüchtigkeitsfehler, der die Entwicklung jedoch kurzzeitig aufhielt.

Eine Schwierigkeit, die beim parallelen Programmieren auftritt, ist der nicht-lineare Ablauf des Programms. Receive-Befehle halten das Programm so lange auf, bis sie die erwartete Nachricht erhalten. Kommt es beim Absender jedoch nicht zum Senden der Nachricht, steckt das Programm fest. In diesem Fall gilt es, das Programm über die Prozessverwaltung zu beenden. Mangels ausreichender Vorerfahrung mit Linux-Systemen und -Servern, musste zuerst in Erfahrung gebracht werden, wie das funktioniert.

Erst relativ spät, bei der Aufbereitung der Ergebnisse, fiel auf, dass einige der errechneten Werte nicht korrekt sein können. Insbesondere ein Maximum von mehr als 365 Tagen im Jahr ließ an der Richtigkeit zweifeln. Generell schienen die Werte im Lauf der Jahre verdächtig groß zu werden. Eine genauere Untersuchung des Problems offenbarte: bei elf Subprozessen waren die ersten elf Jahre korrekt, alles was darauf folgte zu hoch. Es stellte

sich heraus, dass die Subprozesse beim Zählen der Tage mit den Vorjahresergebnissen weiterrechneten – ein Punkt, der im ersten Jahr 36 Tage mit Sauerstoffmangel vorwies, und im zweiten Jahr 17, wurde im zweiten Jahr mit 53 defizitären Tagen angegeben. Das Zurücksetzen der *output*-Matrix nach jedem Durchgang behob das Problem.

Ein bis zu diesem Zeitpunkt nicht behobenes Problem ist die Auswirkung der Programmreihenfolge auf das Ergebnis. Aus Übersichtlichkeitsgründen war angedacht, Teile des Programmcodes für den Hauptprozess im Quelltext unter dem Code für die Subprozesse zu platzieren. Stattdessen stehen nun alle Anweisungen für Rang 0 über denen der Subprozesse, auch wenn sie logisch erst nach Beendigung dieser ausgeführt werden. Als Beispiel ist die abschließende Akkumulation der Werte zu nennen. Wechseln sich Anweisungen für Hauptprozess und Subprozesse ab, erreicht das Programm offenbar nicht alle Anweisungen. Das folgende Code-Beispiel soll das Problem verdeutlichen.

Listing 5.2: If-Verzweigungen mit MPI

```
1 if (rank .eq. 0) then
2     ! Haupt-Anweisung 1
3 end if
4 if (rank .ne. 0) then
5     ! Subanweisung 1
6 end if
7 if (rank .eq. 0) then
8     ! Haupt-Anweisung 2 wird nicht erreicht
9 end if
```

Das Problem wurde umgangen, indem die zweite Haupt-Anweisung unmittelbar nach der ersten folgt.

Literaturverzeichnis

[Gre10] N. Greenwood. Detection of low bottom water oxygen concentrations in the North Sea; implications for monitoring and assessment of ecosystem health. *Biogeosciences*, pages 1357–1373, April 2010.

[Lor12] I. Lorkowski. Interannual variability of carbon fluxes in the North Sea from 1970 to 2006 – Competing effects of abiotic and biotic drivers on the gas-exchange of CO2. *Estuarine, Coastal and Shelf Science*, pages 1–20, 2012.

Abbildungsverzeichnis

Listingverzeichnis

Anhänge

A. Ergebnis-Tabelle

Jahr	Summe	Ausdehnung	Maximum
1970	155	5	69
1971	869	37	91
1972	4376	101	102
1973	2533	107	107
1974	1685	97	92
1975	1799	94	105
1976	3976	130	131
1977	3142	113	119
1978	2415	111	92
1979	3306	117	116
1980	2770	110	110
1981	8236	166	104
1982	5579	161	150
1983	3010	115	101
1984	6734	156	166
1985	1180	43	125
1986	2856	116	115
1987	2969	110	116
1988	2616	99	109
1989	5398	126	115
1990	1317	72	102
1991	3040	113	91
1992	823	44	96
1993	3445	85	112
1994	2605	135	98
1995	2647	125	99
1996	818	43	96
1997	2745	116	112
1998	1188	43	82
1999	5119	138	100
2000	5598	132	101
2001	1167	79	92
2002	8189	160	98
2003	5920	141	107
2004	1044	80	96
2005	2211	82	109
2006	1197	68	98

Tabelle A.1.: Tablle mit allen Ergebnissen

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 17.05.2015 .....