Performanceanalyse der Ein- / Ausgabe des Ökologiemodells ECOHAM5

Masterarbeit

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

Vorgelegt von:	Simon Kostede
E-Mail-Adresse:	kostede@gmail.com
Matrikelnummer:	6053576
Studiengang:	Master of Sience Informatik
Erstgutachter:	Prof. Dr. Thomas Ludwig
Zweitgutachter:	Dr. Michael Kuhn
Betreuer:	Dr. M. Kuhn, F. Große, Dr. H. Lenhart

Hamburg, den 22.08.2016

Abstract

Das Ziel dieser Arbeit ist die Analyse der Ein- und Ausgabe (E/A) des Ökosystemmodells ECOHAM5. ECOHAM5 ist ein paralleles HPC-Programm, das mit MPI parallelisiert ist. Es werden NetCDF Dateien als Ergebnis der Simulation ausgegeben. Wie bei vielen Erdsystem- und Klimamodellen wird bei ECOHAM5 nur serielle E/A durchgeführt, was die Skalierung stark einschränkt. Für die Analyse wurde ECOHAM5 für parallele E/A erweitert und es wurde die Performance gemessen und analysiert. Zudem wurde parallele E/A in ECOHAM5 implementiert.

ECOHAM5 ist ein Erdsystemmodell, das die Ökologie der Nordsee simuliert. Das Modell wird genutzt um Fragen des Kohlenstoffflusses in der Nordsee im Rahmen des Klimawandels (Lorkowski et al., [LPMK12]; Pätsch und Kühn, [PK08]) sowie Fragen zur Auswirkung unterschiedlicher Belastung des Ökosystems Nordsee durch Nährstoffeinträge von Stickstoff und Phosphor (Lenhart et al. [LMBB$^+$10, LDG$^+$13]) zu untersuchen. Dazu wird die Nordsee in ein dreidimensionales Gitter unterteilt und für jede Gitterzelle werden für eine Reihe von Zustandsvariablen numerische Differenzialgleichungen gelöst. Das Modellgebiet des ECOHAM-Gitters umfasst den Nordwesteuropäischen Kontinentalschelf (NECS) und Teile des angrenzenden Nordostatlantiks (Abbildung 4.1). ECOHAM5 ist in Fortran implementiert und nutzt MPI für die parallele Ausführung mit mehreren Prozessen. Jeder dieser Prozesse ist an der Berechnung der Simulation beteiligt. Die Simulationsergebnisse werden in der ursprünglichen Version von ECOHAM5 von einem Prozess/Rechenknoten, dem Masterknoten, mit NetCDF gespeichert. Diese serielle E/A wurde in dieser Arbeit verschiedentlich untersucht. Die Implementierung wurde statisch anhand des Quellcodes analysiert. Die Ausführung wurde gemessen und mithilfe des Tracingprogramms Vampir/Score-P ausgewertet. Für die E/A nutzt ECOHAM5 die Bibliotheken MPI, MPI-IO, HDF5 und NetCDF.

Die neue Version von ECOHAM5 mit paralleler E/A konnte sich, auf dem Testsystem mit 10 Rechenknoten, zeitlich nicht von der Version mit serieller E/A absetzten, sondern war etwa zwischen 15% und 25% langsamer.

Danksagung

Ich möchte mich bei Michael Kuhn, Hermann Lenhart und Fabian Große für die Betreuung der Arbeit und für die Hilfe bei der Implementierung der parallelen I/O bedanken. Auch bedanken möchte ich mich bei Prof. Dr. Thomas Ludwig für die Möglichkeit diese Arbeit zu schreiben.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel werden die Grundlagen der Arbeit erläutert.

Heutige Hochleistungsrechner bzw. High Performance Computing (HPC) Systeme lösen rechenaufwendige Probleme mit einem hohen Grad an Parallelisierung. In dieser Arbeit wird parallele E/A im Programm ECOHAM5 implementiert und die parallele und serielle E/A von ECOHAM5 analysieren. In diesem Kapitel wird ein Überblick über den derzeitigen Stand im Bereich HPC gegeben. Die Hintergründe zu paralleler Ein- und Ausgabe (Input / Output (I/O)) und parallelen Speichersystemen in parallelen HPC-Systemen werden im folgenden Kapitel erläutert. Abschliessend wird die Aufgabenstellung dieser Arbeit dargestellt.

Die Arbeit befasst sich mit der Analyse der E/A des Ökologiemodells ECOHAM5. Insbesondere die parallele E/A mit NetCDF/HDF5/MPI-IO wird betrachtet. ECOHAM5 wurde im Laufe dieser Arbeit auf parallele E/A für die Ein- und Ausgabe umgestellt.

1.1 Hochleistungsrechnen / Wissenschaftliches Rechnen

Hochleistungsrechnen (High Performance Computing (HPC)) bezeichnet einen Bereich in der Informatik, der sich mit dem Betrieb und der Programmierung von Hochleistungscomputern beschäftigt. Hochleistungscomputer bezeichnet dabei Computersysteme, die aus mehreren diskreten Computern bestehen und die über Hochleistungsnetzwerke verbunden sind. Moderne Hochleistungsrechner bzw. Supercomputer bestehen aus vielen einzelnen Computern, sog. Knoten, die oftmals bestimmte Aufgaben innerhalb des Supercomputers übernehmen. Die meisten Konten sind Berechnungsknoten, die mit mehr Prozessorleistung ausgestattet sind und die wissenschaftliche Berechnungen durchführen. Weitere Knoten sind die Speicherknoten, die anfallende Daten auf persistente Medien speichern. Die Anforderungen, die von den Anwendern bzw. Anwendungsprogrammen an Hochleistungsrechner gestellt werden, haben zur Folge, dass extreme Rechenleistungen und Speicherkapazitäten verwendet werden müssen. HPC-Systeme werden überall verwendet, wo rechenintensive Probleme gelöst werden müssen. Beispiele sind in den Naturwissenschaften, in der Kryptographie, in der Erdsystemforschung oder auch im Bereich der Computer Generated Images (CGI) und bei Animationsfilmen zu finden. Moderne HPC-Systeme sind oft als Cluster organisiert, wie beispielhaft in Abbildung 1.1 gezeigt.

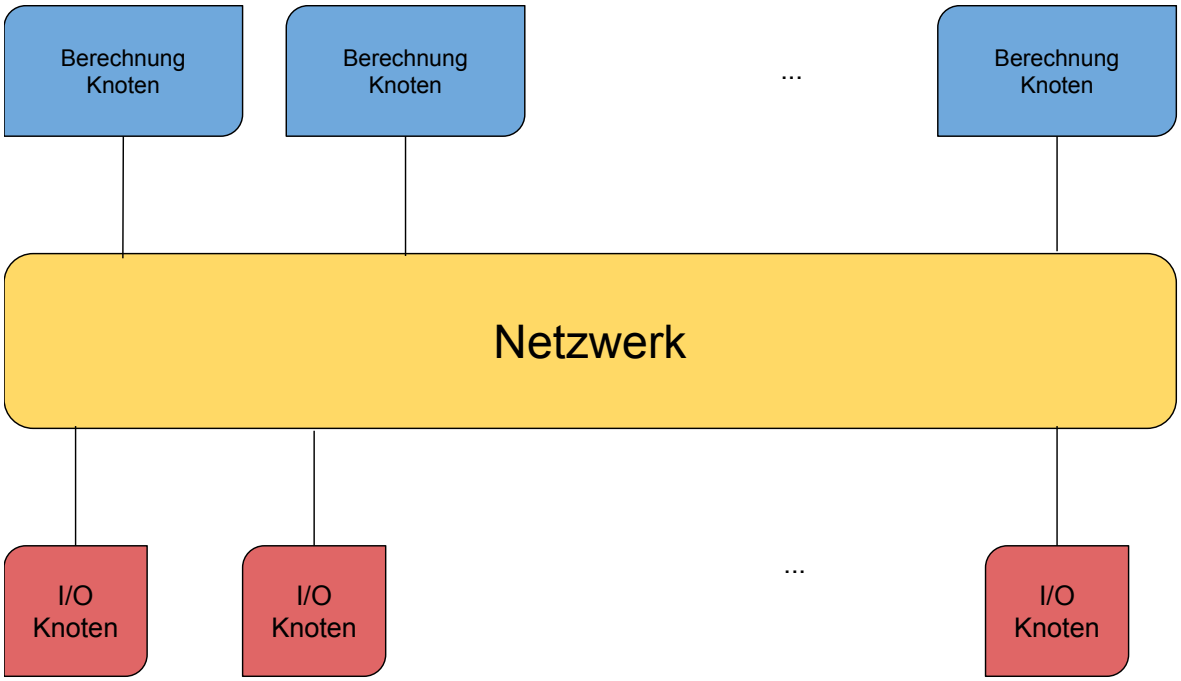

Abbildung 1.1: HPC-System

Dabei werden die Berechnungsknoten (Compute Nodes) über ein Hochleistungsnetzwerk miteinander verbunden. Jeder Knoten hat meist mehrere Prozessoren mit mehreren Prozessorkernen. Auf jedem Knoten wird mindestens eine Instanz eines Computerprogramms ausgeführt, ein Prozess. Jeder Prozess kann aus mehreren Threads bestehen. Ein Thread ist ein eigenständiger Ausführungsstrang innerhalb eines Prozesses (vergl. [Wik16a]). Diese Prozesse nutzen dann die E/A-Knoten und ein Speichersystem um Ein- und Ausgabe durchzuführen. Die Kommunikation zwischen den einzelnen Knoten erfolgt per Nachrichtenaustausch (Message Passing). Als Standard für die Programmierung von Nachrichtenaustausch wurde das Message Passing Interface (MPI) etabliert. Damit können Programmierer portable parallele Programme schreiben, die Nachrichten zum Informationsaustausch zwischen den Knoten oder Prozessen nutzen. Details zu MPI werden in Abschnitt 2.1.1 aufgeführt. Im Kontext des Nachrichtenaustauschs ist auch die Bandbreite des Netzwerks wichtig, da viele Knoten miteinander kommunizieren. Es gibt mehrere verbreitete Netzwerktechnologien, wie Infiniband oder Ethernet. Die maximale Bandbreite von Ethernet ist derzeit 100 GBit/s[1], bis Ende 2017 soll die Datenrate auf 400 GBit/s gesteigert werden[Tro15]. Infiniband[MRP+15] hat in der aktuellen Version EDR eine Bandbreite von 300 GBit/s, diese wird sich mit der Version HDR, die etwa 2017 erscheinen soll, auf 600 GBit/s steigern.

Jedes parallele Programm löst ein Problem bzw. eine Aufgabenstellung. Dieses globale Problem wird in Teilaufgaben aufgeteilt. Jeder Prozess berechnet ein Teilstück der

[1] Si Präfix[Wik16b] Giga = 10^9 Bits

globalen Problemstellung. Zwischen den Knoten werden dann die relevanten Daten der lokalen Teillösung per Nachricht ausgetauscht. Dieser Lösungsansatz ist nicht für jedes Problem angemessen, da Probleme serielle Abhängigkeiten haben können. Daher können nicht alle Probleme voll parallelisiert werden. Formal wird dieser Zusammenhang im Amdahlschen Gesetz [Amd67] beschrieben, das besagt, dass die maximal mögliche Geschwindigkeitssteigerung der Parallelisierung durch $\frac{1}{Ts}$ beschränkt wird. Dabei wird mit T_s der seriellen Anteil der Laufzeit des Programms bezeichnet. Die maximale Geschwindigkeitssteigerung die ein Programm bei einem seriellen Anteil von 1% erreicht werden kann, ist Faktor 100.

Alle aktuellen Erdsystem- und Klimamodelle nutzen derzeit noch serielle E/A. Daher folgt für diese Modelle aus dem Amdahlschen Gesetz, dass sie in der Skalierung eingeschränkt sind. In dieser Arbeit wird die Umstellung eines Erdsystemmodells auf parallele E/A vorgenommen um die Einschränkung zu umgehen. Auf zukünftigen HPC-Systemen wird die Auswirkung dieser Einschränkung, aufgrund der höheren Parallelisierung, noch stärker zum Tragen kommen.

FLOPS die Abkürzung für „floating point operations per second“ steht, also die Gleitkommaoperationen, die pro Sekunde ausgeführt werden können, stellen das übliche Leistungsmaß eines Systems dar. Es ist nicht definiert, welche Operationen auf welchen Daten ausgeführt werden, meistens werden Multiplikationen und Additionen genutzt. Dazu schreiben Dongarra et al. [DLP03]:

> „The performance of a computer is a complicated issue, a function of many interrelated quantities. […] the algorithm, the size of the problem, the high-level language, the implementation, the human level of effort used to optimize the program, the compiler's ability to optimize, the age of the compiler, the operating system, the architecture of the computer, and the hardware characteristics. The results presented for benchmark suites should not be extolled as measures of total system performance […] but, rather, as reference points for further evaluations.“
>
> „Die Performance eines Computers ist ein kompliziertes Thema, eine Funktion von vielen zusammenhängenden Eigenschaften. […] dem Algorithmus, der Problemgröße, der Hochsprache, der Implementation, der menschlichen Anstrengung zur optimierung des Programms, den Möglichkeiten des Compilers zu Optimieren, das Alter des Compilers, das Betriebssystem, der Rechnerarchitektur und der Hardwarecharakteristik. Benchmarkergebnisse sollten nicht zu Messungen der Systemperformance überhöht werden, […], sondern als Referenzpunkte für weitere Evaluationen dienen.“

Eine exakte Spezifikation der FLOPS, die ein HPC-Systems im Regelbetrieb erreicht, ist unmöglich. Es ist möglich die theoretische Maximalleistung eines Systems zu bestimmten. Dieser Wert wird auch als Peak Performance bezeichnet. Er wird ermittelt, indem die theoretische Maximalleistung eines Prozessors auf die Anzahl der Prozessoren im System übertragen wird. Die Performance eines Prozessors hängt dabei stark von seiner Taktrate ab. Peak Performance bildet somit eine obere Grenze für die Rechenleistung

eines HPC-Systems. Dabei ist es unwahrscheinlich, dass eine Anwendung diesen FLOPS-Wert erreicht. Um realitätsnahe FLOPS-Werte zu erhalten werden Benchmarks benutzt. Benchmarks sind standardisierte Programme, die in einer kontrollierten Umgebung ausgeführt werden. Damit die Benchmarkergebnisse auf normale Anwendungsprogramme übertragbar sind, ist es wichtig, dass die Benchmarks Berechnungen ausführen, die auch in den echten Anwendungsprogrammen vorkommen. Im HPC Bereich wird HPC LINPACK als ein de-facto Standard eingesetzt. Lineare Gleichungssysteme kommen in mehreren Zweigen der Naturwissenschaften vor, für verschiedene dieser Gleichungssysteme beinhaltet LINPACK Lösungsroutinen. Die TOP500 Liste [Top16] der leistungsstärksten HPC-Systeme nutzt LINPACK als Benchmark im Bereich von mehreren PetaFLOPS.

1.2 Erdsystemmodellierung

Erdsystemmodellierung beschäftigt sich mit der numerischen Modellierung der Erde. Es werden Modelle erzeugt, die mithilfe eines Hochleistungsrechners ein Erdsystem berechnen können. Die Erdsystemmodelle nutzen dabei auf modernen Hochleistungsrechnern viele diskrete Rechenknoten welche über Hochleistungsnetzwerke miteinander verbunden sind. Das Ausführen von Erdsystemmodellen ist ein Aufgabengebiet von HPC-Systemen. Wie vom DKRZ in [DKR16] beschrieben, wird das zu berechnende Gebiet der Erde in einem Erdsystemmodell in ein dreidimensionales Gitter unterteilt. Dieses Verfahren wird auch als Partitionierung (Partitioning) bezeichnet. Für jeden Gitterknoten werden dabei Werte, wie Temperatur, Druck oder Stickstoffkonzentration berechnet und vorgehalten. Die Berechnung erfolgt dabei in diskreten Zeitschritten. Bei der Berechnung eines Zeitschritts werden meist die Daten der benachbarten Gitterknoten betrachtet. Dieses Verfahren ist sehr gut für die Parallelisierung auf einem HPC-System geeignet, da jeder Prozess einen oder mehrere Gitterknoten berechnet. Auch muss jeder Prozess nur mit wenigen anderen Prozessen kommunizieren, solange das Gitter optimal auf die HPC Knoten verteilt ist. Das Verfahren bezeichnet man als Mapping. Inzwischen gibt es auch Ansätze für die Partitionierung des zu berechnenden Systems, die nicht auf rechteckigen bzw. quadratischen Gittern basieren. So besteht das Gitter des neue Klimamodell ICON (Icosahedral non-hydrostatic) des Max-Planck-Institut für Meteorologie [fM16] aus einem Ikosaeder, der durch Polygone weiter unterteilt wird.

Ökosystem-Modelle sind eine Art von Erdsystemmodellen. Mit ihnen wird ein Ökosystem berechnet. Wie in Abbildung 1.2 zu sehen, wird dabei ein Ökosystem auf seine Grundzüge reduziert und dann in dem Programm abgebildet. Für die Abbildung im Programm werden Gleichungen aufgestellt die das Verhalten des Ökosystems abbilden. Um das daraus entstandene Modell zu validieren, werden die Berechnungsergebnisse mit Messwerten aus dem Ökosystem verglichen.

Für die Berechnung von Erdsystemmodellen gibt es verschiedene Modellierungsansätze. Dabei wird insbesondere zwischen globalen Modellen und regionalen Modellen unterschieden. Bei einem globalen Modell wird die gesamte Troposphäre abgebildet, in einem regionalen Modell nur ein bestimmter geographischer Ausschnitt. Es werden die gleichen Berechnungen vorgenommen, nur können diese bei einem regionalen Modell

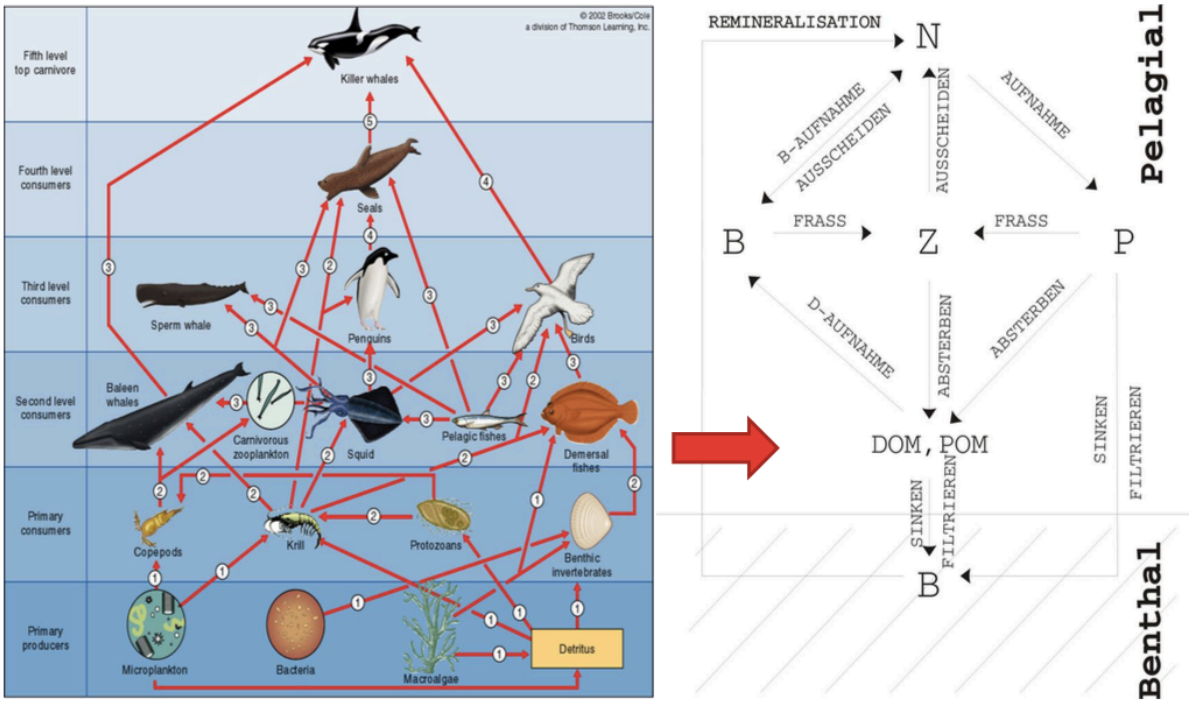

Abbildung 1.2: Modellierung des Ökosystems

zeitlich und/oder räumlich höher aufgelöst werden, da mehr Rechenleistung für kleinere Gebiete oder Zeitschritte aufgewendet werden kann. Nachteil von regionalen Modellen ist, dass Daten für den Rand des Modells vorhanden sein müssen. Diese Daten (Forcing Daten) müssen für den gesamten Berechnungszeitraum vorliegen.

1.3 Parallele E/A

Parallele E/A beschreibt eine Methode für den Zugriff auf Daten aus parallelen Anwendungen heraus, die das Speichern oder Lesen gegenüber der seriellen E/A beschleunigen soll. Dabei wird auf die Daten von mehreren Knoten parallel zugegriffen, anstatt auf die Daten von einem Master-Rechenknoten aus zuzugreifen. Im Fall des seriellen Datenzugriffs ist der Datendurchsatz, im besten Fall, auf den Durchsatz der Anbindung des Master-Rechenknotens an das Speichersystem beschränkt. Eine weitere Einschränkung bei serieller E/A ist, dass viele verteilte Dateisysteme pro Knoten nur einen bestimmten Datenduchsatz bereitstellen können. Dieser Datendurchsatz liegt oft unter dem Durchsatz der Netzwerkschnittstelle. Die Einschränkung wird mit paralleler E/A umgangen. Im optimalen Fall kann nun jeder Knoten gleichzeitig den Datendurchsatz des Master-Rechenknotens erreichen. Im Falle des Forschungsclusters bedeutet das, das eine Anwendung im seriellen Modus maximal 100MB/s in das Lustre schreiben kann. Sollte eine Anwendung auf allen 10 Knoten parallel in das Lustre schreiben, so können bis zu 1000MB/s erreicht werden.

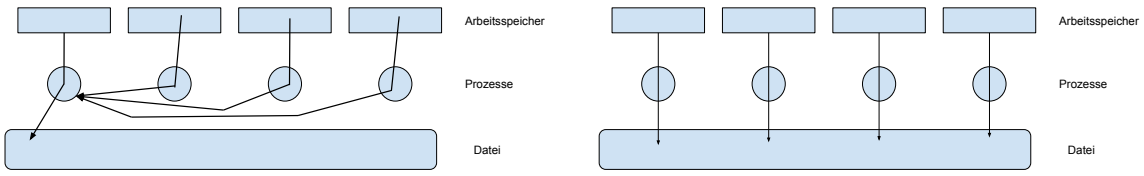

Abbildung 1.3: Parallele E/A und serielle E/A

HPC Anwendungen speichern Daten normalerweise in standardisierten Formaten. So können die Daten einfach ausgetauscht und weiterverarbeitet werden. Das Hierarchical Data Format 5 (HDF5) und das Network Common Data Form 4 (NetCDF) sind Datenformate und Softwarebibliotheken, die eine Programmierschnittstelle (API) für die das Schreiben, Manipulieren und Lesen von Anwendungsdaten bereitstellen. Die Bibliotheken unterstützen Anwendungen, die in verschiedenen Sprachen geschrieben sind, wie C oder Fortran. Beide Bibliotheken sind in den Naturwissenschaften weit verbreitet. NetCDF und HDF5 werden im Abschnitt 2.2 und im im Abschnitt 2.3 Detail erläutert.

HPC Anwendungen sind in vielen Fällen datenintensive Anwendungen. Die Gesamtperformance einer Anwendung ist daher stark abhängig von der E/A Performance. Ein Beispiel davon ist der Programmstart und die Initialisierung mit den Eingabedaten. Bei einem parallelen Programm kann die E/A trotzdem seriell erfolgen. Dann erfolgt der Zugriff auf die Daten über einen einzelnen Knoten, meist den Master-Knoten, der die Daten über MPI erhält bzw. an die anderen Knoten verteilt. Diese Art von serieller E/A ist noch häufig in HPC-Programmen vorhanden. In Programmen mit serieller E/A werden die Daten mittels MPI auf dem Masterknoten gesammelt, wenn sie weggeschrieben werden. Dieser Arbeitsschritt entfällt bei paralleler E/A. Die beiden Vorgehensweisen sind in Abbildung 1.3 zu sehen. Die Parallelisierung der E/A auf mehrere Prozesse kann die Zeit, die für E/A aufgewendet wird, deutlich reduzieren. Bei paralleler E/A ist die theoretische Maximalbandbreite die Summe der Prozessbandbreiten. Jedoch kann die praktische Bandbreite von vielen Faktoren beeinflusst werden und stark unter die Maximalbandbreite absinken. In HPC-Systemen werden die Festplatten über das Netzwerk angesprochen, so dass die maximale E/A-Bandbreite durch das Netzwerk begrenzt wird. Auch die Zugriffsmuster der Anwendungen haben einen starken Einfluss auf die E/A-Bandbreite. Der zufällige Zugriff auf Speichermedien ist bei allen aktuellen Speichermedien langsamer als der sequentielle Zugriff. Sollte die Anwendung Daten so abrufen, dass viele zufällige Zugriffe entstehen, verringert sich die Bandbreite der E/A teilweise stark.

Um parallele E/A durchführen zu können, wird ein paralleles Dateisystem benötigt. Ohne dieses Dateisystem würden die E/A beim Zugriff auf das Dateisystem wieder serialisiert. Parallele Dateisysteme speichern die Daten nicht auf einer Festplatte, sondern verteilen diese auf verschiedene Speicherknoten mit vielen Festplatten. Dabei wird auch der parallele Zugriff auf die Datei von mehreren Prozessen ermöglicht. Es gibt mehrere parallele Dateisysteme, zum Beispiel OrangeFS oder Lustre.

Sowohl HDF5 als auch NetCDF-4 ermöglichen den parallelen Zugriff auf Daten. Dabei bilden die Bibliotheken eine Schicht in einem hierarchischen E/A-Modell. Diese Struktur

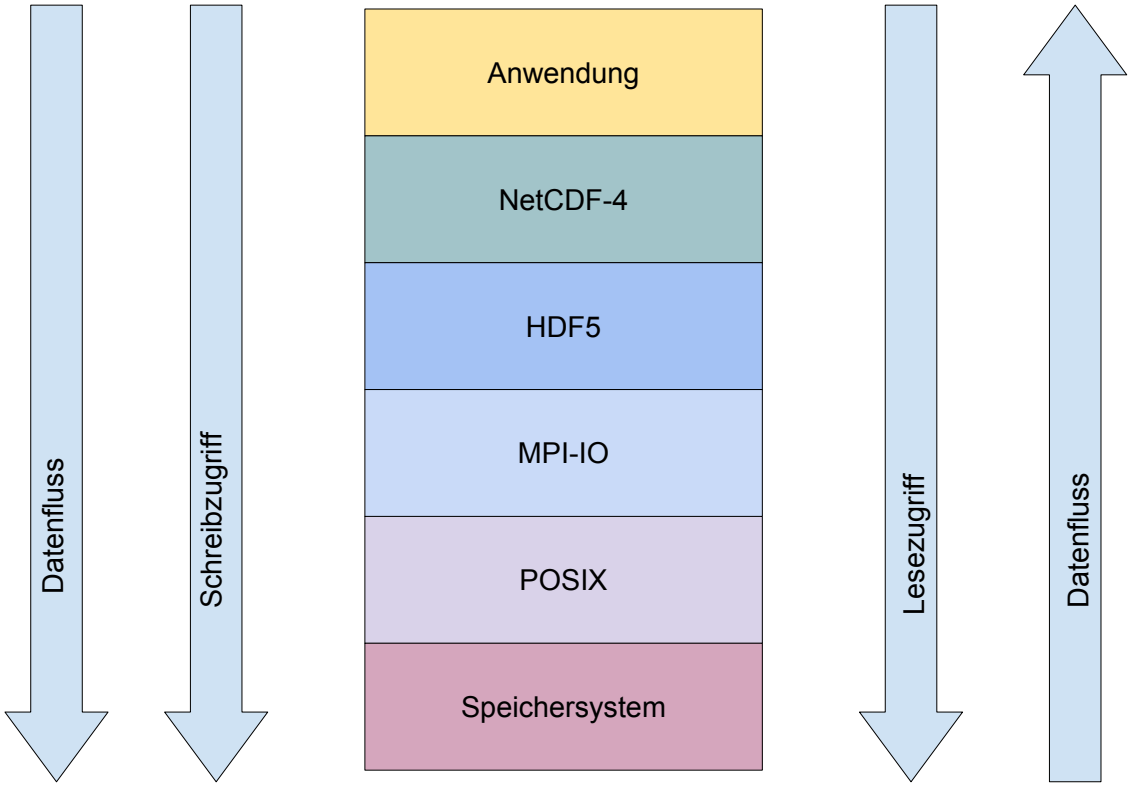

Abbildung 1.4: Hierarchisches E/A Modell

ist in Abbildung 1.4 zu sehen. So nutzt NetCDF-4 HDF5, um Daten zu speichern. HDF5 wiederum greift auf die E/A-Routine von MPI zu. MPI nutzt, abhängig vom verurteilten Dateisystem, entweder den POSIX[2] Standard um die Daten zu speichern oder einen eigenen E/A-Treiber für spezielle verteilte Dateisysteme z.B. OrangeFS. POSIX wird vom Betriebssystem implementiert und dort werden die Daten an den E/A Treiber übergeben, welcher sie auf die Festplatte speichert. Schreibt eine Anwendung Daten, benutzt sie die höchste Ebene der E/A-Hierarchie. Die Daten werden dann durch die Schichten nach unten gegeben, bis sie beim E/A Treiber ankommen. Leseoperationen laufen umgekehrt ab.

Das Ziel dieser Arbeit ist es die parallele E/A im Ökosystem-Modell ECOHAM5 mit den erwähnten Bibliotheken, NetDF-4 und HDF5, einzuführen und zu evaluieren. Die Evaluation betrachtet den kompletten Datenfluss durch die Speicherhierachie. Dabei sollen Probleme und Engpässe bei der praktischen Verwendung von paralleler E/A gefunden und identifiziert werden.

Diese Arbeit nutzt Lustre als paralleles Dateisystem. Lustre wird von der Arbeitsgruppe Wissenschaftliches Rechnen bereitgestellt, wo diese Analyse stattfindet. Zudem ist es unter einer Open Source Lizenz veröffentlicht und wird, nach OpenSFS [Ope16] auf

[2]Portable Operating System Interface

> „[…]seven of the TOP10 supercomputing sites, and over 70% of TOP100 sites[…]“
>
> „[…]sieben der TOP10 Supercomputer und über 70% der TOP100 Supercomputer[…]“

eingesetzt. Die signifikante Verbreitung von Lustre macht es sinnvoll ECOHAM5 für die Analyse darauf auszuführen.

2 Hintergrund

In diesem Kapitel werden die Technologien, die für diese Arbeit benutzt wurden, erläutert.

2.1 MPI

Wie in Kapitel 1 angeführt, nutzt ein typisches HPC-System Nachrichtenaustausch um zwischen den einzelnen verteilten Prozessen zu kommunizieren. Das Message Passing Interface (MPI) ist der Standard um portable parallele Programme zu erstellen, die Nachrichtenaustausch nutzen. MPI definiert dafür eine Schnittstelle, deren Funktionen von verschiedenen Hochsprachen, wie C, C++ oder Fortran, genutzt werden können. Der folgende Abschnitt basiert auf Gropp et al. [WG99]. Nach Gropp et al. [WG99] wurde MPI in zwei Phasen, von einem offenen Forum aus Herstellern von Supercomputern, Autoren von Bibliotheken und Anwendungsentwicklern entworfen. Dieses Forum wird MPI-Forum genannt. Die erste Entwurfsphase fand zwischen 1993 und 1994 statt und ergab die erste Veröffentlichung von MPI, die MPI-1 genannt wird. Eine Reihe von wichtigen Punkten wurde in MPI-1 bewusst nicht behandelt um die Entwicklung und Veröffentlichung zu beschleunigen, so Gropp et al. [WG99]. Ab 1995 traf sich das MPI-Forum erneut um kleiner Korrekturen und Änderungen an MPI-1 vorzunehmen. In 1997 wurde dann MPI-2 veröffentlicht. Die großen Neuerungen in MPI-2 sind parallele E/A (MPI-IO), Remote Memory Operations und dynamisches Prozessmanagement. Es gibt mehrere Implementationen von MPI, beispielsweise MPICH [MPI16] oder OpenMPI [Ope]. Diese Implementationen sind auf einer Vielzahl von Prozessorarchitekturen verfügbar. Der MPI Standard erlaubt es effiziente Implementierungen auf verschiedenen Architekturen vorzunehmen, da nur die Logik einer Operation vorgegeben wird und nicht wie eine Operation ausgeführt werden sollte. Zum Beispiel haben manche Architekturen einen Kommunikations-Coprozessor, auf den eine MPI Implementierung die Kommunikation verlagern kann.

Wie erwähnt, ist die Portierbarkeit von Anwendungen eine Kernanforderung an MPI. Das heißt, Anwendungsprogrammierer müssen ihre Programme nicht für andere Rechnerarchitekturen anpassen. Des weiteren ermöglicht MPI auch das transparente Verwenden von verschiedenen Rechnerarchitekturen durch ein Programm, etwa wenn Teile des Rechensystems auf einer anderen Rechnerarchitektur basieren (vergl. Gropp et al. [WG99]). Die Implementierung transformiert dann die Nachrichten transparent zwischen den Architekturen. MPI fordert von allen Implementierungen ein gewisses Grundverhalten ab. Dazu zählt beispielsweise, dass die Nachrichtenübermittlung zuverlässig ist.

Ein MPI Programm ist während der Ausführung eine Menge von einzelnen Prozessen. Jeder Prozess führt das Programm unabhängig in seiner eigenen Umgebung aus. Die

Prozesse kommunizieren über die MPI Schnittstelle. Während der Ausführung hat jeder Prozess normalerweise seinen eigenen Adressraum, aber es ist auch möglich mit geteiltem Speicher zu arbeiten. Zum Organisieren von Prozessen bietet MPI das Konzept Group (Gruppe) an. Jeder Prozess hat in seiner Gruppe eine eindeutige Identifizierung, den Rank (Rang). Communicators werden zur Kommunikation innerhalb einer Gruppe (intracommunicator) oder zur Kommunikation mit einer anderen Gruppe genutzt (intercommunicator). So können verschiedene Communicators genutzt werden um Teile eines MPI-Programms voneinander fachlich zu trennen. Bei der Trennung von zwei Teilen können z.B. die Communicators COMM_MET und COMM_OZEAN definiert werden, welche die Kommunikation für ein meteorologisches Modell und ein ozeanographisches Modell ermöglichen. MPI definiert einen standardmäßigen Communicator, MPI_COMM_WORLD, welcher alle Prozesse eines MPI Programms enthält. Er löst auch das Problem, dass für die Erzeugung eines Communicators ein Communicator benötigt wird.

Die Basis der Kommunikation via Nachrichtenaustausch von MPI bildet die Punkt-zu-Punkt-Übertragung. Dabei werden Daten direkt von einem Prozess, dem Sender, an einen anderen Prozess, den Empfänger, übertragen. Damit das Anwendungsprogramm keine manuelle Transformation vornehmen muss, kann die Übertragung mit Informationen zur automatischen Transformation von Daten in heterogenen Systemen versehen werden. Außerdem kann ein Tag, in Form eines Integer, an die Nachricht angebracht werden. Diese Markierung erlaubt es dem Empfänger die Nachricht individuell einzuordnen. Die Kommunikationsroutinen von MPI ermöglichen sowohl die Verwendung von blockierender als auch nicht-blockierender Kommunikation. Im blockierenden Modus kehrt die Methode nur nach erfolgreichem Abschluss der Funktion zurück. Im nicht-blockierenden Modus kehrt die Routine sofort zurück. Der Anwender muss dann später manuell überprüfen, ob der Funktionsaufruf erfolgreich war. Das Absenden einer Nachricht führt nicht zwingend zu ihrer Zustellung. Dafür muss sich der Empfänger im Empfangsmodus befinden. Um die Koordination zwischen Sender und Empfänger zu vereinfachen, bietet MPI den synchronen Modus. Dabei koordinieren sich Sender und Empfänger und die Sendeoperation ist nur erfolgreich, wenn der Empfang vom Empfänger bestätigt wurde.

Als weiteren Kommunikationsmodus stehen in MPI die kollektiven Kommunikationsoperationen (collective communication) zur Verfügung. Diese ermöglichen die Kommunikation zwischen allen Prozessen, die einer (intracommunicator-)Gruppe angehören. Weitere Kommunikationsfunktionen sind Scatter, Gather. Bei Scatter werden Daten von einem Quellsystem (root) auf alle Prozesse der Gruppe verteilt, dabei erhält jeder Prozess einen etwa gleichgroßen Anteil der Daten. Gather ist die umgekehrte Funktion. MPI erlaubt auch den Broadcast von Daten, wobei eine Nachricht von einem Prozess an alle anderen Prozesse in der Gruppe gesendet wird. Andere Operationen erlauben die verteilte Reduktion und Berechnung von Daten. So können z.B. Summen effizient von mehreren Prozessen berechnet werden. Für die Synchronisation zwischen verschiedenen Prozessen enthält MPI das Konzept einer Barrier. Der Funktionsaufruf von `MPI_Barrier` blockiert, bis alle Prozesse in der Gruppe den Aufruf durchgeführt haben.

2.1.1 MPI-IO

MPI-IO ist eine Erweiterung des MPI-Standards, der spezifiziert, wie parallele E/A mit MPI durchzuführen ist. Wie erwähnt, wurde MPI-IO erst nachträglich in MPI-2 eingeführt. MPI-IO bietet dem Entwickler viele Operationen um E/A aus einem parallelen Programm herraus vorzunehmen. MPI-IO sieht verschiedene Möglichkeiten vor, um parallele E/A in einer Anwendung zu benutzten. Im einfachen Fall kann eine Datei mit MPI parallel geöffnet werden, damit die Anwendung lesend oder schreibend zugreifen kann. Da bei dieser Art des Zugriffs keine Koordination der E/A durch MPI erfolgt, muss die Anwendung diese selber lösen. Wenn jeder Knoten einen bestimmten zusammenhängenden Teil der Daten einliest, handelt es sich um continuous Zugriffe. Als non-contiguous wird der Zugriff auf nicht zusammenhängende Teile der Datei bezeichnet, wenn ein Knoten z.B. jeden 10. Datenblock einliest. MPI unterstützt auch die Nutzung von non-contiguous Zugriffen durch Anwendung, in dem es „File Views“ unterstützt. Dabei kann ein Knoten nur Teile der Datei sehen und darauf lesend und schreibend zugreifen.

MPI-IO trennt E/A in zwei Modi, Independet und Collective. Im Independet Modus führt jeder Prozess die E/A ohne Koordinierung oder Synchronisierung durch. Im Collective Modus wird diese Koordinierung durchgeführt, wodurch Optimierungen möglich werden, beispielsweise der Einsatz von Two-Phase-I/O.

Wie auch MPI selber ist MPI-IO ein Standard für den verschiedene Implementationen existieren. Die verbreiteste Implementierung ist ROMIO [Lab16]. ROMIO wird von verschiedenen MPI Implementierungen genutzt, zum Beispiel OpenMPI oder Cray MPI, es ist aber inzwischen Teil von MPICH. Thakur et al.[TGL99] erläutern die zwei Kernoptimierungen, die in ROMIO bei E/A-Operationen vorgenommen werden. Die Optimierungen sind Data Sieving und Two-Phase-I/O. Data Sieving nutzt aus, dass mehrere unabhängige und zufällige Zugriffe auf eine Festplatte, wegen des Seek-Overheads, aufwendiger sind als ein umfassender zusammenhängender Zugriff. Daher werden viele unabhängige Zugriffe aggregiert und gemeinsam ausgeführt. Im Zuge dieses zusammenhängenden Zugriffs werden auch Bereiche geladen, welche nicht angefordert wurden. Diese Bereiche werden dann im Client Anwendungspuffer gefiltert (sieving). Die zweite Optimierung Two-Phase-I/O greift, wenn Collective-I/O eingesetzt wird. Diese Optimierung basiert auch auf der Minimierung von kleinen unabhängigen Zugriffen. Die Optimierung betrachtet das gesamte Zugriffsmuster und identifiziert die sich überschneidenden Zugriffe. Für diese Zugriffe wird die E/A Operation an einen Aggregator delegiert. Dort werden die Zugriffe zu einem großen gemeinsamen Zugriff zusammengefasst, wie in Abbildung 2.1 dargestellt. Die Daten werden dann reorganisiert an die einzelnen Client Prozesse übertragen, so dass jeder Prozess nur seine Daten erhält.

2.2 HDF5

Hierarchical Data Format (HDF) bezeichnet eine Sammlung von selbstbeschreibenden Dateiformaten und Bibliotheken mit einer E/A Schnittstelle um auf diese Dateien zuzugreifen. HDF wird von der „The HDF Group“ vorangetrieben und entwickelt. Die HDF

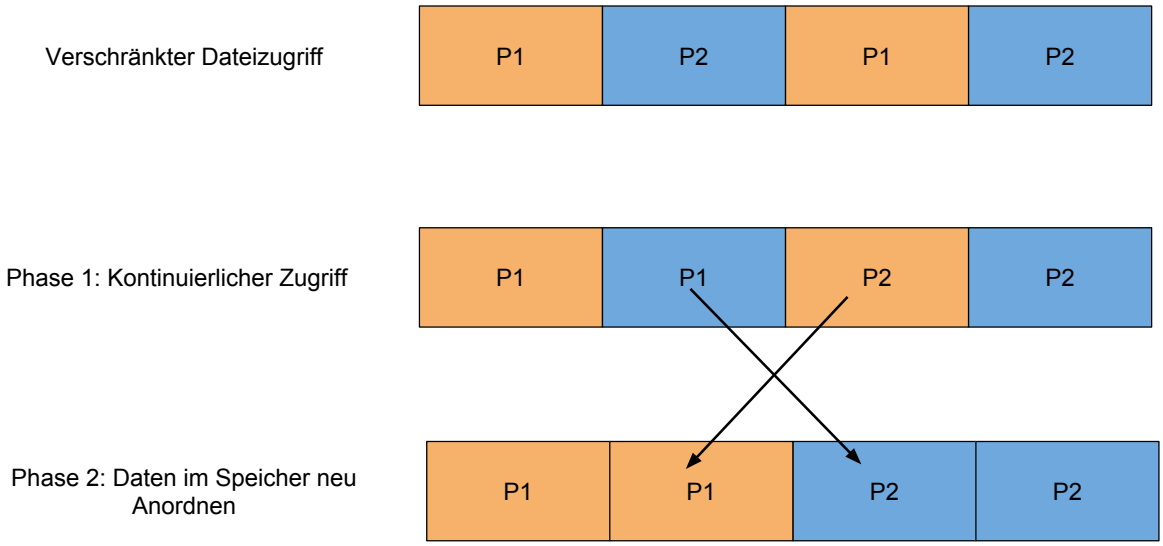

Abbildung 2.1: Zwei Phasen E/A

Group unterstützt derzeit das alte HDF4 Format und das neue HDF5 Format. HDF4 wurde aufgrund seiner Einschränkungen abgelöst. Insbesondere die maximale Dateigröße von 2 GiB ist für fast alle Anwendungsbereiche im Hochleistungsrechnen nicht mehr angemessen. Dieser Abschnitt basiert auf dm HDF User Guide [Gro16].

Implementierungen von HDF5 existieren für verschiedene Sprachen, z.B. C, C++, Fortran und Java . HDF5 unterstützt heterogene und große Daten. Es kann eine unbegrenzte Anzahl von Daten und Objekten gespeichert werden. Diese können auch aus einem beliebigen Datenmix bestehen. Komplexe Datentypen, Abhängigkeiten und Relationen können auch abgebildet werden. Es ist möglich nur einen Teil der Datei einzulesen, wodurch der Zugriff beschleunigt wird. Die Analyse, Verarbeitung und Verwaltung von HDF5 Dateien wird durch viele open-source und proprietären Programmen ermöglicht.

HDF5 Dateien unterteilen sich in zwei Oberdatentypen, `Datasets` und `Groups`. Datasets sind homogene multidimensionale Arrays eines einzigen Datentyps. Sie können mit Dateien im Dateisystem verglichen werden. Die Metadaten eines Datasets werden in einem Header gesichert. Der Header enthält den Namen, den Datentyp, Datenlayout (Dataspace) und Speicherlayout des Datasets. Jedes Element im Dataset hat den gleichen Datentyp, dieser kann entweder elementar (Integer, Float, …) sein oder zusammengesetzt. Das Datenlayout beschreibt die Struktur des Arrays, welche Dimension es hat, was die maximale Dimension ist und wie groß es ist. Die Anzahl von Dimensionen in einem Dataset ist begrenzt. Eine Dimension kann als unbegrenzt definiert werden. Das Dataset kann mit einer Collective-I/O Operation bis zur maximalen Größe der Dimension vergrößert werden.

Eine Group ist ein hierarchischer Datentyp, welcher Datasets oder andere Groups enthält, vergleichbar mit einem Ordner im Dateisystem. Beide Oberdatentypen können mit Attributen versehen werden. Diese erlauben es Metadaten hinzuzufügen. Mit dieser hierarchischen Datenstruktur ist es möglich die `POSIX` Pfadsemantik auf HDF5 zu über-

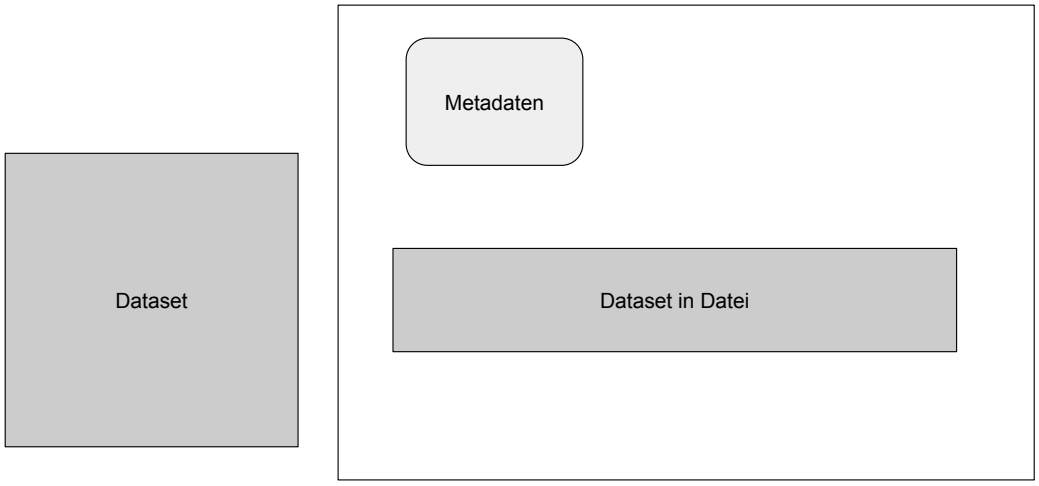

Abbildung 2.2: Kontinuierliches Dataset in HDF5-Datei (vergl. [Gro16])

tragen und z.B. per „`/pfad/zu/daten`“ auf Ressourcen zuzugreifen. Auf den Groups und Datasets basierend werden alle komplexeren Datenstrukturen aufgebaut, zum Beispiel die Schnittstellen für Bilder oder Tabellen.

Daten können in HDF5 in drei Modi gespeichert werden, entweder kontinuierlich, kompakt oder in Blöcken. Der erste Modus, kontinuierlich, bedeutet, dass das Datenset, abgesehen vom Header, als ein ununterbrochenes Byte-Array gespeichert wird, wie in Abbildung 2.2 zu sehen. Mehrdimensionale Datasets werden, wie sie im Arbeitsspeicher abgelegt sind, zu einem 1-dimensionalen Array serialisiert. Daraus folgt auch, dass die Daten beispielsweise bei Fortran mit der Spalte zuerst gespeichert werden.

Auch muss das Dataset eine feste Größe haben, da nicht garantiert werden kann, das freier Platz hinter dem Array zur Verfügung steht.[1] Der zweite Modus, kompakte Datenspeicherung, bedeutet, dass die Daten direkt im Header gespeichert werden. Hierbei werden die Daten zusammen gespeichert, im Gegensatz zu den anderen Modi. Dies ist nur sinnvoll möglich, wenn die Datenmengen klein sind. Das Dataset muss auch hierbei eine feste Größe haben. In diesem Modus können Header und Daten in einer zusammenhängenden E/A Operation geladen werden. Die blockweise Speicherung ist der dritte Modus. Die Daten werden hierfür in Blöcke aufgeteilt. Blöcke sind rechteckige Regionen des Dataset. Die Blöcke werden unabhängig an beliebige Orte der Datei geschrieben, wie in Abbildung 2.3 dargestellt.

Blöcke können aus beliebigen Unter-Dimensionen des Datasets bestehen. So kann ein 3-dimensionales Dataset in 3-, 2- oder 1-dimensionale Blöcke unterteilt werden. Die Position der Blöcke in der HDF5 Datei werden im Header gespeichert. B-Bäume werden genutzt um die Blockpositionen effizient zu speichern und auffindbar zu machen. Der

[1] „Contiguous storage is the simplest model. It has several limitations. First, the dataset must be a fixed-size: it is not possible to extend the limit of the dataset or to have unlimited dimensions.“ [Gro16]

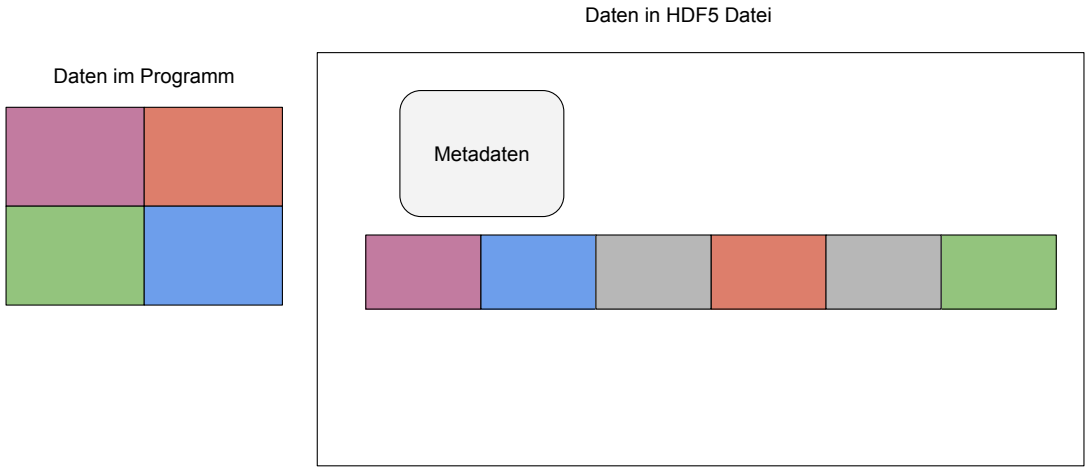

Abbildung 2.3: Chunked Dataset in HDF5-Datei (vergl. [Gro16])

Zugriff auf B-Bäume erfolgt in logarithmischer Zeit, sowohl für Lesen als auch Verändern. Solche Bäume werden auch in Dateisystemen wie BTRFS oder in Datenbanken benutzt.

HDF5 verarbeitet Blöcke einzeln bei E/A-Operationen. Blöcke werden in einem Cache vorgehalten. Auf Blöcke wird immer komplett zugegriffen, auch wenn nur Teile der Daten angefordert wurden, und dabei in den Cache geladen. Die angeforderten Daten werden in den Anwendungsspeicher übertragen. Blöcke ermöglichen die transparente Kompression in HDF5. Kompression ist nur mit Blöcken möglich, da die Kompressionsfunktionen auf Blöcke angewendet werden. Zusätzlich hat auch jedes Dataset einen Cache. Der Cache wird von der Anwendung kontrolliert und kann abgeschaltet werden. Die Abschaltung des Caches ist beim Einlesen der gesamten Datei sinnvoll.

HDF5 nutzt verschiedene Strategien um Speicherplatz zu allozieren und mit Fülldaten (fill values) vorzubelegen. Dies ist besonders im parallelen Betrieb notwendig, damit lesende Prozesse feststellen können, ob ein Bereich eines Datasets schon von einem schreibenden Prozess geschrieben wurde. Standard bei HDF5 ist ein Füllwert (fill value) von 0, wie in POSIX `read()`:

> „If any portion of a regular file prior to the end-of-file has not been written, read() shall return bytes with value 0“ [IG13]

Die Schnittstelle erlaubt es eigene Füllwerte zu setzen. Weiterhin existieren drei Modi zum Anlegen bzw. Allozieren von Datasets und zur Vorbelegung mit Füllwerten, „Early“, „Late“ und „Incremental“ [Gro16]. Im Modus „Early“ wird das Dataset bei der Erstellung komplett alloziert. Im Modus „Late“ wird das Dataset alloziert, wenn das Dataset geschrieben wird. Nur mit Blöcken ist der Modus „Incremental“ möglich. Dabei wird die Allokation beim Schreiben des Blocks vorgenommen. Wenn kein Modus gesetzt wird, versucht HDF5 den besten Modus abhängig von der Speichermethode und Zugriffsmethode zu wählen, z.B. abhängig davon, ob Parallele E/A benutzt wird. Das

Schreiben von Füllwerten kann auch komplett abgeschaltet werden. Beim Schreiben von Füllwerten wird die Datei doppelt geschrieben, einmal mit dem Füllwert und einmal mit den echten Daten. Die Auswirkungen auf die Performance sind erheblich. Das Verhalten sollte, wenn möglich, deaktiviert werden.

2.3 NetCDF

Network Common Data Form (NetCDF) ist ein standardisiertes, selbstbeschreibendes Dateiformat für den Austausch von wissenschaftlichen Daten und eine Bibliothek um auf diese Daten lesend und schreibend zuzugreifen. Dieser Abschnitt basiert auf [RDE+16]. Das Format basiert historisch auf dem NASA-Format CDF, ist zu diesem aber nicht mehr kompatibel. NetCDF wird von der University Corporation for Atmospheric Research (UCAR) vorangetrieben, eine alternative Version wird von Unidata entwickelt. Es handelt sich um ein selbstbeschreibendes binäres Format. Ein Header beschreibt das Layout der Datei, unter anderem um welche Endianess es sich handelt, wie die Data-Arrays aussehen und diverse Metadaten. NetCDF gibt es in drei Hauptversionen, das klassische Standartlayout (NetCDF classic), ein erweitertes Layout mit 64bit Offsets (NetCDF large) und einem neuen Layout, das auf HDF5 basiert, welches mit NetCDF Version 4 (NetCDF-4) eingeführt wurde. Das neue Layout ermöglicht Dateien, deren Größe nur vom Dateisystem abhängt und mehrere unbegrenzte Dimensionen in der Datei. Zusätzlich gibt es eine Abspaltung von NetCDF, PnetCDF (parallel NetCDF), die als erste parallele E/A implementiert. PnetCDF hat keine zu NetCDF vollständig kompatible Schnittstellen und schreibt ein eigenes Format.

Diese Arbeit setzt sich mit NetCDF-4 auseinander.

NetCDF wird in den Naturwissenschaften eingesetzt, besonders in den Bereichen Klimaforschung, Wetter, und Ozeanographie. Ähnlich zu HDF5 werden die Daten in Arrays gespeichert. In einem Array werden Elemente eines Datentyps gespeichert. Arrays können multidimensional sein. Die Schnittstelle von NetCDF abstrahiert mit abstrakten Datentypen von der konkreten Datenspeicherung. Die interne Repräsentation von Daten wird von NetCDF gekapselt, da der Anwender nur über die Schnittstelle der Bibliothek auf die Daten zugreift. Diese Methodik erlaubt es NetCDF portable zu sein und unabhängig von der Rechnerarchitektur zu sein. Die Implementierung kann auch getauscht werden. Es gibt Implementierungen für verschiedene Programmiersprachen und Umgebungen, wie C, C++, Fortran 77 / 90 oder Java. Des weiteren sind Implementierungen für verschiedene BSD/UNIX Systeme und auch Microsoft Windows verfügbar.

Das klassische Datenmodell von NetCDF basiert auf Datasets. Ein NetCDF-Dataset besteht aus einer Dimension, Variablen und Attributen. Diese werden sowohl mit einem Namen als auch einer ID-Nummer versehen. Bei NetCDF-4 wurden Gruppen eingeführt. Jede NetCDF Datei enthält eine standardmäßige (root) Gruppe. Jede Gruppe kann eine oder mehrere Untergruppen oder konkrete benutzerdefinierte Datentypen enthalten, wie in Abbildung 2.4 zu sehen. Jede Gruppe bildet dabei ein klassisches NetCDF-Dataset ab und hat jeweils eigene Attribute, Variablen und Dimensionen. Dimensionen können bei NetCDF beim Erstellen der Datei oder im „define“ Modus definiert werden. Seit

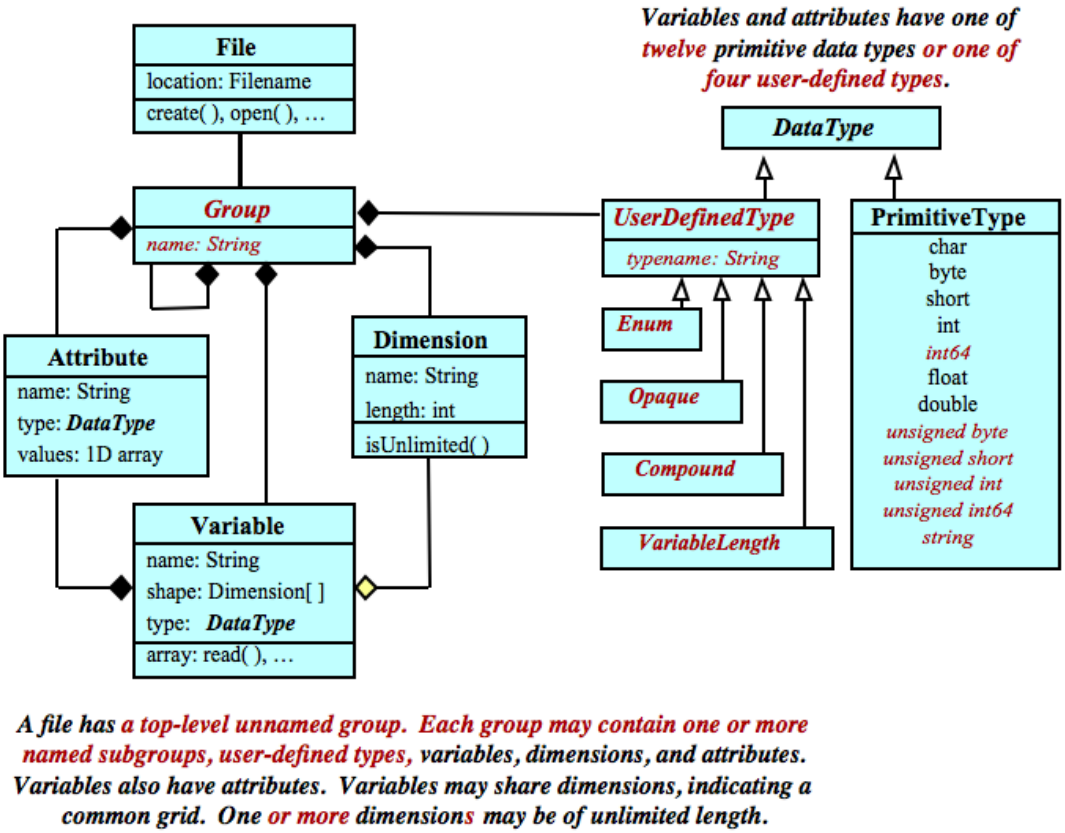

Abbildung 2.4: NetCDF-4 Datenmodell nach [RDE+16]

NetCDF-4 können mehrere Dimensionen pro Datei unbegrenzt sein. Die Anzahl an Dimensionen pro Gruppe ist unlimitiert, allerdings hat NetCDF ein Soft-Limit bei 1024 Dimensionen, unter anderem um die Entwicklung von Tools zu vereinfachen. Dieses Limit kann durch das Anheben von `NC_MAX_DIMS` umgangen werden. Die Dimensionen definieren die Form der Gruppe. Die meisten Daten in NetCDF werden in Variablen gespeichert. Eine Variable repräsentiert ein Array von Werten desselben Datentyps. Eine Variable hat einen Namen, einen Typ und eine Form, welche von den bei der Variablenerstellung angegebenen Dimensionen festgelegt wird. Attribute, die der Variablen zugeordnet sind, können jederzeit hinzugefügt oder geändert werden. Variablen, die mindestens eine unlimitierte Dimension haben, werden als „record variable“ bezeichnet und sie können eine unbegrenzte Anzahl von Einträgen enthalten. Wenn die Variable keine Dimension enthält, kann nur ein Wert gespeichert werden. Solche Variablen werden als Scalar bezeichnet, Variablen mit einer Dimension als Vektor und Variablen mit zwei Dimensionen als Matrix. Diese Namenskonventionen gleichen denen von Fortran oder denen der Mathematik.

Als Koordinaten-Variable (Coordinate Variables) werden Variablen bezeichnet, die den gleichen Namen wie eine Dimension haben. Außerdem ist eine solche Variable ein

Vektor und die Form ist identisch zu der Form der Dimension. Solche Variablen haben für NetCDF keine besondere technische Bedeutung, aber sie sind Teil der NetCDF Namenkonvention und werden von NetCDF-Tools und Software besonders behandelt. Analog zum Namen werden solche Variablen benutzt um physikalische Koordinaten aus einem Koordinatenreferenzsystem in der dazugehörigen Dimension abzubilden. Bei ECOHAM5 gibt es beispielsweise die Variablen Latitude, Longitude und Depth. Diese Variablen sind Teil der Namenkonvention von NetCDF. Die Variable Depth hat zum Beispiel 31 Ebenen. Jeder dieser Ebenen ist ein bestimmter Tiefenwert, etwa 5 Meter, zugeordnet. Ein Tool wie ncview kann bei der Visualisierung der Ausgabedatei mit den entsprechenden zugeordneten Tiefenwerten arbeiten und diese anzeigen.

In NetCDF wird zwischen dem internen und externen Datentyp einer Variable unterschieden. Die Datentypen der Programmiersprache des Anwendungsprogramms sind interne Datentypen. Externe Datentypen sind Typen, die von der NetCDF Schnittstelle bereitgestellt und innerhalb der NetCDF Datei genutzt werden. Ab NetCDF-4 werden die Datentypen, sofern notwendig, automatisch umgewandelt. Die Vorteile von externen Datentypen sind die Portabilität, da Anwendungsentwickler sich nicht um das Speichersystem kümmern müssen. Zusätzlich kann NetCDF die Definition der Typen ändern ohne dass Anwendungsprogramme geändert werden müssen, da die Umwandlung in der Bibliothek stattfindet.

Attribute werden in NetCDF für Metadaten oder andere ergänzende Daten genutzt. Das System ist vergleichbar zu den Schemadefinitionen in Datenbanken. Attribute, die Informationen über die gesamte Datei bereitstellen, werden globale Attribute genannt. Die meisten Attribute enthalten Informationen zu Variablen. In NetCDF-4 können Attribute auch Gruppen zugeordnet werden. NetCDF hat reservierte Attribute, die für die Programmierschnittstelle genutzt werden. Außerdem gibt es eine Reihe von standardmäßig genutzten Attributen, beispielsweise das globale Attribut `title`, das eine kurze Beschreibung des Datasets enthalten sollte. Da es verschiedene Konventionen für Attribute gibt, ist das Attribut `Metadata_Conventions` von besonderer Wichtigkeit. Attribute sollten nur genutzt werden um Metadaten festzuhalten. Es können keine Attribute für Attribute definiert werden, daher sollten Werte, die Metadaten benötigen, als Variablen definiert werden. Attribute können nur Skalar- oder Vektordaten enthalten, keine mehrdimensionalen Daten. Zusätzlich sollten die Daten in einem Attribut klein genug sein um dauerhaft im Arbeitsspeicher verbleiben zu können.

HDF5 wird, wie oben erläutert, von NetCDF für die Datenspeicherung genutzt. Standardmäßig wird das kontinuierliche Datenlayout für Nicht-Record-Variablen genutzt. Das Block Datenlayout kann vom Anwender ausgewählt werden. Bei Record Variablen wird das Blocklayout genutzt, da nur dieses die notwendige Erweiterbarkeit bietet. Das Schreiben von Füllwerten kann über NetCDF unterbunden werden.

Für NetCDF gibt es verschieden generische Software-Werkzeuge. Diese können entweder NetCDF-Dateien visualisieren oder verarbeiten. Ein Werkzeug, ncdump, kann eine NetCDF-Datei einlesen und für Menschen lesbar auf der Konsole ausgeben. Das Gegenstück von ncdump ist ncgen. ncgen kann, aus einer für Menschen lesbaren Beschreibungssprache, eine binäre NetCDF-Datei erzeugen. Diese Beschreibungssprache heißt CDL (network Common Data form Language). netCDF Operators (NCO) [NCO16] ist

ein Sammlung von Werkzeugen um, vor allem in geographischen Koordinatensystemen verortete, NetCDF-Dateien zu analysieren. Beispiele sind das Ermitteln von neuen Daten oder die Berechnung von Mittelwerten. Die Ergebnisse der Analysen können als Text-, Binär- oder als NetCDF-Datei ausgegeben werden.

2.4 Lustre

Lustre ist ein freies paralleles Dateisystem, das vor allem im wissenschaftlichen Hochleistungsrechnen eingesetzt wird. Es wird über eine POSIX kompatible Schnittstelle angesprochen. Die Schnittstelle ermöglicht es auf die verteilten Dateiobjekte parallel zuzugreifen. Dieser Abschnitt basiert auf [Int16b]. Lustre speichert Daten objektbasiert. Eine Datei wird in mehrere Objekte / Streifen aufgeteilt (`stripes`), welche dann auf unterschiedlichen Speicherservern gespeichert werden. Die Verteilung auf verschiedene Server ermöglicht die Performance des parallelen Dateisystems. Die E/A-Bandbreite des Speichersystems setzt sich aus der Summe der Bandbreiten der Speicherserver zusammen. Die Bandbreite eines einzelnen Servers wird vom Netzwerk und der Zugriffsbandbreite der Festplatten begrenzt. Der Gesamtspeicherplatz setzt sich aus dem Speicherplatz der einzelnen Speicherserver zusammen.

Die Architektur von Lustre besteht aus drei Teilen. Diese Teile sind Lustre Clients, Metadata Server (MDS) und Object Storage Server (OSS). Abbildung 2.5 zeigt diese einzelnen Komponenten. Clients verbinden sich über das Netzwerk mit den Lustre Komponenten für Metadaten und Datenspeicher. MDS halten die Metadaten des Dateisystems vor, dazu zählen Dateinamen, Ordner, Zugriffsrechte und Dateilayout. Diese Metadaten werden vom MDS auf den Metadata Targets (MDT) gespeichert. Der Dateiinhalt wird auf den Object Storage Targets (OST) gespeichert, auf die über ein OSS zugegriffen wird. Dabei wird eine Datei auf mehrere OSS verteilt. Typischerweise werden zwei bis acht OST auf von einem OSS angesprochen und auf diesem OSS-Knoten ausgeführt. Clients greifen über die POSIX Schnittstelle parallel auf die Dateien zu.

Der parallele Zugriff auf eine Datei kann die Performance steigern. Dazu muss die Gesamtbandbreite, die für eine Datei zur Verfügung steht, die Bandbreite eines einzelnen OSS übersteigen. Die Größe eines Streifens ist die Streifengröße (stripe size). Die Anzahl der genutzten OSTs je Streifen ist der Parameter Streifenanzahl (stripe count). Wenn beim Schreiben eines Datenblocks auf einen OST die Streifengröße überschritten wird, wird der nächste Block in einen neuen Streifen auf einem anderen OST geschrieben. OSTs werden nach dem Verfahren Round Robin ausgewählt. Standardmäßig werden Streifengrößen 1 MiB und eine Streifenanzahl von 1 benutzt, also keine Nutzung von mehreren Servern je Streifen. Diese Werte können pro Datei und pro Verzeichnis angepasst werden.

Beim Zugriff auf eine Datei können die Clients selbständig ermitteln mit welchem OST zu kommunizieren ist, da die Streifen Round Robin verteilt werden. Es ist keine Kommunikation mit dem MDS notwendig wenn das Layout einer Datei bekannt ist.

Lustre setzt auf einen Mechanismus von verteilten Sperren um die Integrität einer Datei sicherzustellen. Die verteilten Sperren erlauben es den Clients Daten lokal zwischen-

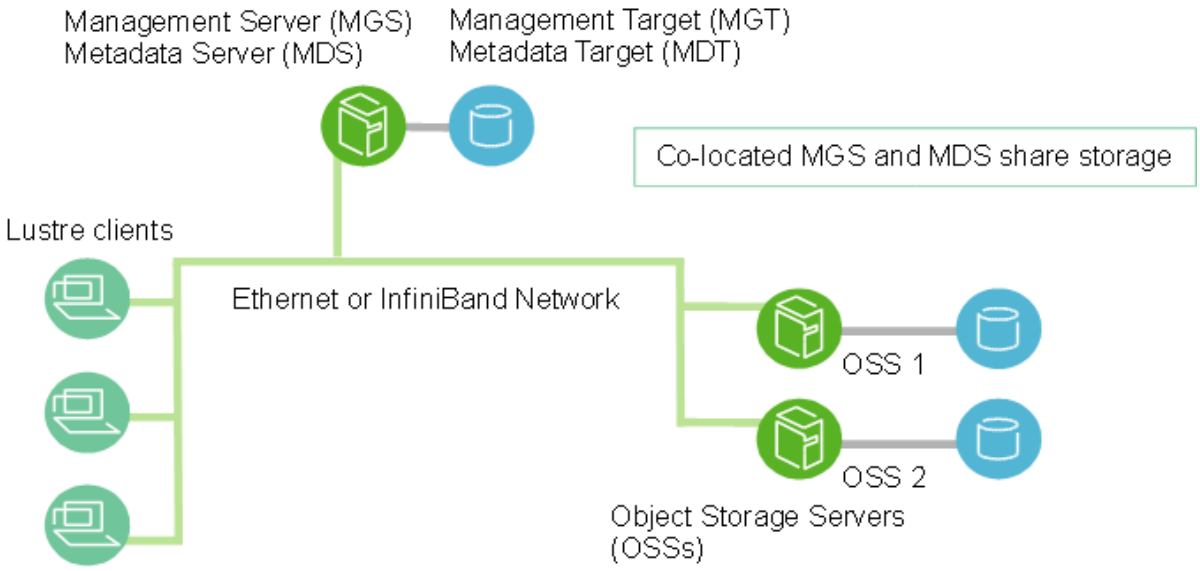

Abbildung 2.5: Lustre Architektur [Int16b]

zuspeichern, da die Sperre parallele Änderungen verhindert. Sperren über Dateiinhalte werden von den OSTs für ihre Dateiregionen verwaltet. Die Sperren werden für byte-weise spezifizierte Abschnitte verteilt. Sperren über Metadaten werden von dem MDT verwaltet, auf welchem die Inode der Datei liegt.

2.5 Vampir und Score-P

Bei der Entwicklung von parallelen Anwendungen ist es oft notwendig mögliche Performance-Engpässe und Flaschenhälse zu entdecken. VampirTrace ist ein Werkzeug für die Analyse und insbesondere Leistungsmessung von verteilten Programmen. Score-P ist der Nachfolger von VampirTrace. Die Analyse der gesammelten Daten findet in dem Werkzeug Vampir statt. Der folgende Abschnitt basiert auf [KHKS08]. Im Allgemeinen können zwei Arten der Analyse unterschieden werden, Profiling und Tracing. Beim Profiling werden Informationen über Ereignisse (Events) zusammengefasst. Beim Tracing werden Informationen über Ereignisse gespeichert. Tracing ist eine mächtigere Analysemethode als Profiling, da aus den Tracing-Daten Profiling Daten erzeugt werden können, aber nicht umgekehrt. Der Nachteil von Tracing ist der hohe Overhead, der dabei entsteht, der die Ergebnisse verfälschen kann. Übliche Ereignisse, die aufgezeichnet werden, sind:

- Ein- und Austritt in Funktionen bei Funktionsaufrufen
- Punkt zu Punkt Kommunikation mit MPI
- Collective Kommunikation mit MPI
- Messungen von Leistungsmetriken

Die Änderung der Anwendung um Ereignisse aufzuzeichnen wird als Instrumentierung bezeichnet. Vampir und Score-P unterstützten die Instrumentierung und das Tracing von Anwendungen. Die Auswertung erfolgt im Vampir-GUI. Vampir/Score-P arbeitet dabei in drei Schritten. Im ersten Schritt wird der Programmcode mithilfe des Vampir-Compilers oder des Score-P Compilers kompiliert. Dieser fügt Instrumentierungs-Code in das Programm ein und kompiliert es dann mit dem eigentlichen Compiler, z.B. mpif90. Im zweiten Schritt werden während der Ausführung des Programms mit dem Instrumentierungs-Code verschiedene Metriken über das eigentliche Programm erhoben. Dabei wird auch die Zeit gemessen, die in den Instrumentierungsfunktionen verbracht wird. Diese Daten werden in Trace-Dateien geschrieben und können im dritten Schritt mit dem Vampir GUI-Programm ausgewertet werden. Dabei erzeugt Vampir otf (Open Tracing Format) Dateien und Score-P otf2 Dateien.

Was Vampir/Score-P zusätzlich von den verbreiteten Tracing-Programmen, wie z.B. pperf, unterscheidet, ist, dass es Tracing-Daten nicht nur auf einem Knoten zu einem Prozess sammelt, sondern dass Vampir und Score-P die Tracingdaten von einem gesamten parallelem Programm aufzeichnen. Dazu synchronisiert Vampir/Score-P die Daten mit hoher zeitlicher Auflösung, damit auch Kommunikationsvorgänge richtig in den Daten abgebildet werden. Die Auflösung ist auf jedem System unterschiedlich und wird in der Trace-Datei angegeben. Auf dem Forschungscluster wird eine Genauigkeit von 375 Picosekunden angegeben.

Die Trace-Dateien können mit dem Tool Vampir geöffnet werden. Es ermöglicht das Explorieren der Daten über Zeitleisten und die Anzeige von statistischen Daten. Vampir bietet verschiedene Auswertungen an:

- Rechenzeit
- Kommunikation
- E/A

Rechenzeit Vampir/Score-P misst die Ausführungszeiten von einzelnen Funktionen innerhalb eines parallelen Programmes. Die Zeiten werden in den verschidenen Charts dargestellt, gruppiert und summiert. Dadurch kann schnell ein Überblick verschafft werden, welche Operationen besonders hohe Anteile an der Gesamtausführungszeit haben.

Kommunikation Es werden von Vampir/Score-P alle MPI-Kommunkationen zwischen verschiedenen Rechenknoten aufgezeichnet. Diese werden in der Auswertung als direktionale Pfeile dargestellt, z.B. zeigen bei MPI-Broadcast Pfeile vom Sender an alle Empfänger.

E/A Vampri/Score-P zeichnet standardmäßig alle MPI-IO Operationen auf. Optional können auch alle POSIX-E/A Operationen aufgezeichent werden. In Vampir wird dann die Dauer der Operationen angezeigt und nach Art aufgeschlüsselt.

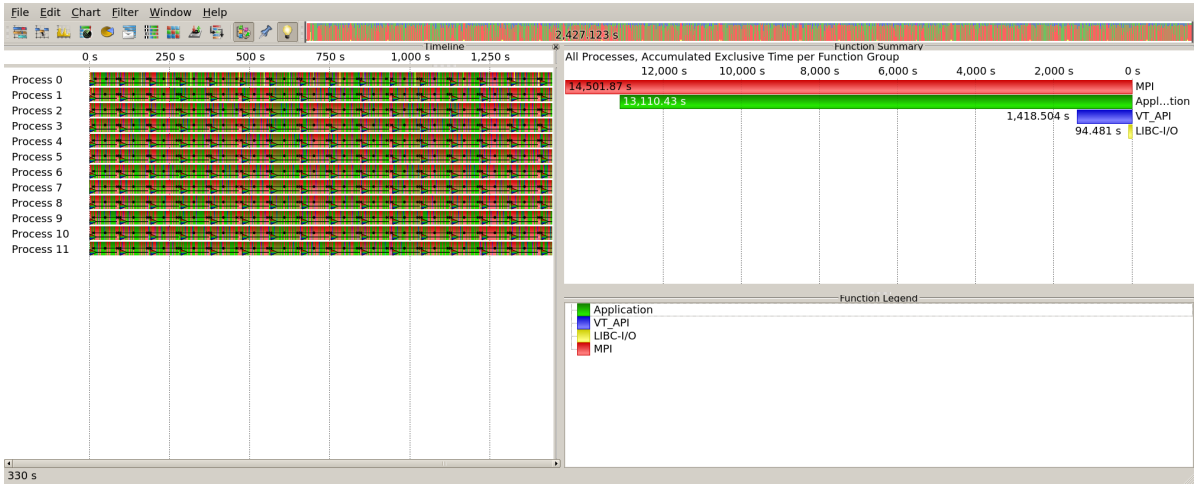

Abbildung 2.6: Vampir Benutzerinterface

Auswertung Vampir hat ein graphisches Benutzerinterface, welches für die Auswertung der Tracing-Daten genutzt wird. Das Tracing erzeugt mehrere komprimierte Dateien, die die Tracingdaten enthalten, und eine Beschreibungsdatei mit der Dateiendung „*.otf*“, die mit Vampir geöffnet wird. Daraufhin lädt Vampir alle Tracingdaten in den Arbeitsspeicher und zeigt sie in der Form, die in Abbildung 2.6 zu sehen ist. In der zentralen Zeitachse werden die Prozesse und Threads vertikal dargestellt. Der aktuelle Zustand des Programms wird über Farben kodiert und mit horizontalen Balken, welche die zu diesem Zeitpunkt aktive Funktionen darstellen, visualisiert. Punkt zu Punkt und globale Kommunikation wird über Pfeile dargestellt.

Nachträgliche Filter Vampir unterstützt das Filtern der Tracing-Daten nach verschiedenen Kriterien. Ein Beispiel ist das Filtern nach Methodenname. Mittels Filter können nur relevante Informationen dargestellt werden. Sie beeinflussen alle Visualisierungen des Programmablaufs. Dabei werden Informationen, die nicht den Filtern entsprechen, visuell versteckt und in die Statistiken nicht einberechnet.

2.5.1 Filter

Bei der Instrumentierung und dem Tracing eines kompletten Programms können große Datenmengen anfallen. Diese Analysedaten können zu groß für die weitere Verarbeitung sein. Umfassende Analysedaten können sehr viel Arbeitsspeicher beanspruchen, so dass sie nicht mehr mit einem Rechner verarbeitet werden könne. Sehr große Datensets können alternativ mit VampirServer geladen werden. VampirServer ist ein verteiltes Programm, das auf mehreren Rechenknoten gestartet werden kann. Vampir kann dann mit dem VampirServer verbunden werden. Weiterhin benötigen alle Operationen in Vampir mehr Zeit auf großen Datensets. Zusätzlich verfälscht der Overhead das Tracing, auch wenn Vampir/Score-P versucht diese Einflüsse zu minimieren. Aus diesen Gründen kann

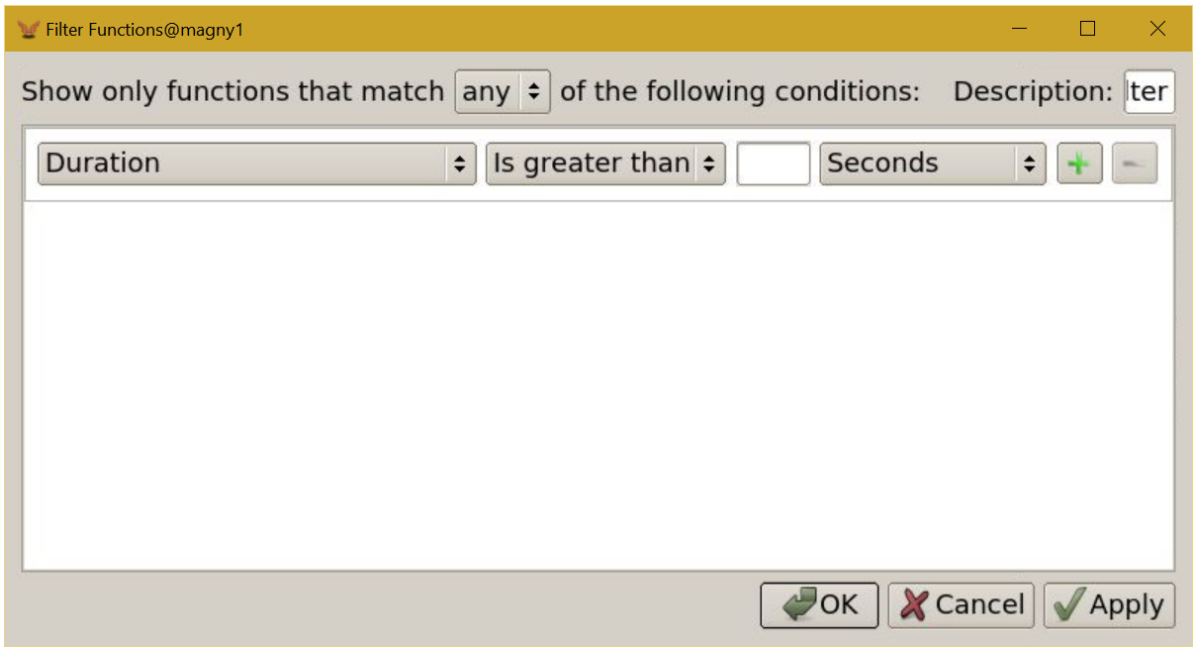

Abbildung 2.7: Vampir Filter Benutzerinterface

es sinnvoll sein die Tracing-Daten schon beim Tracing einzuschränken. Dies kann über Filter erreicht werden. Vampir/Score-P ermöglichen diese Art der Filterung. Nachteil kann dabei die Verfälschung der relativen Laufzeit der instrumentiert Funktionen im Vergleich zu nicht instrumentierten Funktion sein. Es besteht die Möglichkeit, das die Daten nicht mehr repräsentativ für einen normalen Programmablauf sind. Dabei gibt es zwei Ansätze für die Wahl der Filter. Inklusive oder exklusive Filter. Für inklusives Filtern gilt, dass nur Methoden oder Subroutinen aufgezeichnet werden, die in der Filterdatei explizit definiert sind. Beim exklusiven Filtern werden Methoden oder Subroutinen, die in der Filterdatei definiert sind, ausgeschlossen. Bei Score-P werden solche Filter in Filter-Dateien definiert, wie im Quellcode 2.1 zu sehen.

```
1 $ cat scorep.filter
2 SCOREP_REGION_NAMES_BEGIN
3  EXCLUDE
4  foo*
5  bar*
6 SCOREP_REGION_NAMES_END
```

Listing 2.1: Beispiel Score-P Filter Datei

Um die Filter-Datei zu laden, muss diese, wie in Quellcode 2.2 dargestellt, als Umgebungsvariable exportiert werden.

```
1 setenv SCOREP_FILTERING_FILE scorep.filter
```

Listing 2.2: Laden der Filter Datei

Von den Score-P Entwicklern wird empfohlen, erst einen Programmablauf nur mit Profiling durchzuführen. Aus diesen Profiling Daten kann dann ein passender Filter ermittelt werden. Mit diesem Filter kann dann ein Programmablauf mit aktiviertem Tracing durchgeführt werden.

2.6 Tools

Es wurden verschiedene Tools für diese Arbeit eingesetzt. Dazu zählen ncview, nccmp und ncdump.

2.6.1 ncview

ncview (vergl. [Pie16]) ist ein Tool, mit dem NetCDF Dateien visualisiert werden können. Die Dateien werden zweidimensional angezeigt, wie in Abbildung 5.7 zu sehen. Weiterhin können die verschiedenen Dimensionen, zum Beispiel die Zeit, der Datei schrittweise durchgegangen werden.

2.6.2 nccmp

nccmp ist ein Tool um NetCDF Dateien zu vergleichen. Nccmp beschreibt sich selbst mit:

> „nccmp compares two NetCDF files bitwise, semantically or with a user defined tolerance (absolute or relative percentage). Parallel comparisons are done in local memory without requiring temporary files. Highly recommended for regression testing scientific models or datasets in a test-driven development environment.“ [Zie15]
>
> „nccmp vergleicht zwei NetCDF Dateien bitweise, semantisch oder mit benutzerdefinierten Toleranzen (absolut oder in Prozenten). Parallele Vergleiche werden, ohne temporäre Dateien, im Arbeitsspeicher durchgeführt. Sehr empfehlenswert für Regressionstests von wissenschaftlichen Modellen oder in testgetriebenen Umgebungen.“

nccmp ist vergleichbar mit dem Unix Tool cmp, welches Dateien bitweise überprüfen kann. Wie im Zitat dargestellt, wurde nccmp für das Testen der neuen E/A Implementierung genutzt.

2.6.3 ncdump

ncdump ist ein Tool, das NetCDF Dateien menschenlesbar ausgeben kann. Dafür kann ncdump die NetCDF Datei insbesondre im Format cdl ausgeben. Zusätzlich kann es verschiedene Informationen über eine NetCDF Datei anzeigen. Zu diesen Informationen zählen die Version der NetCDF Datei oder z.B. die Metadaten einer NetCDF Datei. ncdump ist Teil von NetCDF.

„The ncdump command-line utility converts netCDF data to human-readable text form. It's useful for browsing data.“ [Uni11]

3 Stand der Wissenschaft

In diesem Kapitel wird auf wissenschaftliche Arbeiten mit Bezug zu dieser Arbeit eingegangen. Viele wissenschaftliche Arbeiten behandeln Ein- und Ausgabeprobleme und parallele E/A im HPC-Bereich. Dabei wird oft der Fokus auf die Optimierung von Software-Bibliotheken oder Middleware-Software gelegt.

Bartz et al. [BCK+15]zeigen, wie verschiedene E/A-Muster die Performance beeinflussen und definieren „Best Practices“. Dabei überprüfen Bartz et al. NetCDF, NetCDF mit Alignment-Patch und HDF5 mit folgenden Konfigurationen, auf einem Lustre Dateisystem,

1. Verschieden Zugriffsmuster: Disjoint und Interleaved
2. Mit und ohne Alignment
3. Collective- und Independent-I/O
4. Continous und Chunked Datenlayout

Für HDF5 empfehlen Bartz et al. Continous Datenlayout, Alignment zu Lustre-Stripes, Independent-I/O, kein Chunked-I/O. Wenn Chunked-I/O notwendig ist, dann sollte es im Disjoint-Zugrifssmuster ausgeführt werden. Bei NetCDF wird insbesondere das Alignment hervorgehoben, für das Bartz et al. einen Patch für NetCDF entwickelt haben, damit E/A mit Aligment in NetCDF möglich wird. Die Optimierungen überprüfen Bartz et al. auf einem HPC-System mit zehn Client Servern und zehn Storage Serveren. Diese sind mit einem 1Gbit/s Ethernet vernetzt, so dass eine maximale E/A Bandbreite von 1.125MB/s gegeben ist. Die Ergebnisse werden von Bartz et al. auch auf größere System anwendbar angesehen.

Bartz [Bar14] geht in seiner Masterarbeit noch genauer auf die Themen des Papers [BCK+15] ein. Das Paper basiert auf der Masterarbeit. In der Arbeit werden insbesondere die gewählte Methodik und der Hintergrund genauer dargestellt. Dabei wird darauf eingegangen, wie die Testfälle entworfen wurden und welche Änderungen an NetCDF vorgenommen wurden. Die Ergebnisse sind sonst analog zu dem Paper [BCK+15].

Howison et al. [How12] zeigen Optimierungen für HDF5 auf einem Lustre-Dateisystem auf. Dabei zeigen sie den Einfluss des Aligments von HDF5-Chunks mit dem darunterliegenden Lustre-Dateisystem auf die Gesamtleistung der E/A-Operationen. Neben dem Aligment, zeigen sie auch auf, das es für eine bestmögliche Auslastung des E/A-Systems notwendig ist, die Metadatenspeicherung auf möglichst wenige E/A-Operationen einzustellen. Dazu zeigen sie zwei Optimierungen. Die erste Optimierung ist das Einpassen des Metadata B-Trees in eine bestimmte Anzahl von Chunks. Die zweite Optimierung ist das Vermeiden von unnötigen Metadaten E/A-Operationen zur Laufzeit, indem die Cache-Eviction der Metadaten in HDF5 von der Anwendung kontrolliert wird. Weiterhin stellen sie einen verbesserten Algorithmus für MPI-IO CB2 vor, welcher das Collective-Buffering von MPI-IO auf das Lustre-Dateisystem anpasst, indem der Algorithmus möglichst ein One-to-One Mapping zwischen Aggregatoren und E/A-Servern nutzt und die E/A-Operationen zu den Lustre-Stripes alignt. Howison et al. halten zu dem CB2 Algorithmus fest, dass er, wenn viele kleine Datensegmente geschrieben werden, aufgrund des Kommunikations-Overheads langsamer sein kann, als der originale Algorithmus. Howison et al. überprüfen ihre Optimierungen anhand von drei HPC-Anwendungen auf drei HPC-Systemen, die bis zu 40960-fach parallel sind. Dabei beschleunigt sich die E/A um das bis zu 33-fache, dabei bleibt die Ausgabedatei eine valide HDF5 Datei.

Isaila et al. [IGC+14] plädieren für eine Neuausrichtung des E/A-Stacks in HPC Anwendungen. Dies machen sie an der steigenden Datenmenge fest, welche für wissenschaftliche und ingenieurwissenschaftliche Anwendungen benötigt werden, insbesondere im Kontext von exascale Systemen. Im Rahmen dieser Betrachtung beschreiben Isaila et al., dass zukünftige Systeme im exascale Bereich mehr Sprachenhierarchien als derzeitige Systeme besitzen werden. Unter diesen Voraussetzungen schreiben sie, dass es nicht mehr ausreichend sein wird, jeweils eine Schicht der Speicherhierarchie einzeln zu optimieren, zu skalieren und zu verbessern. Stadtessen schlagen sie vor, das Speichersystem in seiner Gesamtheit zu verbessern. Besonders Technologien wie NVRAM[1] werden die Speicherhierarchie vertiefen. Isaila et al. identifizieren drei Problemkategorien: (i) Mangelhafte Programmierbarkeit des Speichersystems bei steigender Tiefe der Speicherhierarchie (ii) mangelnde Koordinierung im Speichersystem im Fall von unerwarteten Ereignissen wie Fehlern oder rapiden Änderungen im Datenvolumen (iii) HPC-Speichersysteme im exascale Bereich machen es notwendig die Lokalität von Daten im System und in den Speicherhierarchien auszunutzen und dem Anwendungsprogramm aufzuzeigen. Dies ist allerdings bei heutigen Speicherinterfaces wie POSIX nicht möglich. Weitere Verbesserungsmöglichkeiten sehen Isaila et al. beim Energieverbrauch der Speichersysteme. Sie beobachten dabei, dass der Energieverbrauch für das Bewegen der Daten in einem HPC-System inzwischen den Energieverbrauch für die Berechnung übertroffen hat. Insgesamt sprechen sich Isaila et al. für neue Ansätze im E/A-System aus, mit besonderem Augenmerk darauf, dass die Lösungen über verschiedene Schichten hinweg arbeiten. Zudem sollten diese Änderungen baldmöglichst vorgenommen werden, um den Übergang zu Exascale-Systemen zu erleichtern.

[1] Nonvolatile RAM

Byna et al. dokumentieren die parallele E/A, die für die Simulation von zwei Billionen Partikeln, die etwa 30 TiB pro Zeitschritt erzeugt, notwendig ist. Dabei zeigen sie, wie sie diese Datenmenge mithilfe von Optimierungen und einem HDF5 Wrapper, H5Par, performant behandeln können. Byna et al. zeigen dabei auch, wie sie von ihrem vorherigen Vorhersagemodell, eine Datei pro MPI Domäne zu speichern, auf eine kontinuierliche HDF5 Datei gewechselt sind. Laut Byna et al. ist der Vorteil von einer Datei pro MPI-Domäne, dass ein großer Teil des maximalen E/A-Durchsatzes des Speichersystems einfach erreicht werden kann. Diesen Ansatz bezeichnen sie als file-per-process (fpp). Mit dem file-per-process-Ansatz werden auch Nachteile in Kauf genommen, so Byna et al., da bis zu 20.000 Dateien pro Zeitschritt erzeugt werden. Außerdem werden die nachfolgenden Schritte der Datenverarbeitung eingeschränkt. Die Daten müssen in einem postprocessing Schritt verarbeitet werden, um die meisten Analysetools benutzen zu können. Die HDF5 Datei wird von Byna et al. mit H5Par geschrieben, dabei vereinfacht H5Par das Verwalten einer großen Anzahl an Partikeln. H5Par kapselt dabei, so Byna et al., viele der Komplexität von HDF5 unter Einbuße der Flexibilität und der beliebigen Datenmodelle von HDF5. Byna et al. war es einfach möglich ihr Modell VPIC auf H5Par umzustellen. Der Wrapper öffnet die Datei automatisch mit den notwendigen bzw. den optimalen Hints und Einstellungen für HDF5 und Lustre. Die Datei wird von H5Par im Modus „MPI-IO Collective Buffering“ geöffnet. Byna et al. beschreiben, wie dabei das Speichern von Daten in zwei Phasen unterteilt wird. In der ersten Phase werden die Daten auf einem Subset der MPI-Tasks aggregiert und gepuffert um dann in der zweiten Phase auf den E/A-Servern gespeichert zu werden. So wird, nach Byna et al., der Wettstreit reduziert, da weniger Knoten mit den E/A Servern kommunizieren. Weiterhin halten Byna et al. fest, dass die von ihnen genutzte Cray MPI-IO durch Alignment das Lustre Dateisystem optimal auslasten kann, wie auch schon von Howison et al. beschrieben.

Son et al. [SSL+13] demonstrieren einen Mechanismus um die Fluktuationen bei der Nutzung von verteilten Dateisystemen und Speichersystemen zu reduzieren. Dieser Mechanismus erlaubt es Son et al. das Subset an Speicher-Server dynamisch zu wählen, welche am geringsten ausgelastet sind, um den bestmöglichen E/A Durchsatz zu erreichen. Um dieses Ziel erreichen zu können nutzen Son et al. einen E/A Software Layer, der es ihnen erlaubt die Datei dynamisch zu partitionieren. Im Zuge dessen demonstrieren Son et al. das die E/A Performance auf geteilten E/A Ressourcen von Fluktuationen der E/A Performance betroffen sein können, bedingt durch Wettstreit um Ressourcen. Des weiteren schlagen Son et al. eine dynamische Überwachung der E/A Bandbreite von einzelnen E/A Servern vor um langsame E/A Server vom File-Striping auszuschließen. Um dies zu erreichen schlagen Son et al. zudem eine transparente Datei-Partitionierung und einen Datei-Layout Transformationsmechanismus vor. Der Mechanismus teilt die Daten auf die E/A Knoten auf. Mit dieser Lösung konnten Son et al. bei bis zu 8.192 Prozessen signifikante Verbesserungen der E/A Performance zeigen. Der gewählte Ansatz kann laut Son et al. wirksam den Einfluss von langsamen E/A-Servern vermeiden und die Schreib-Performance steigern. Nach Son et al. wird für die dynamische Überwachung der E/A

Bandbreite im ersten Schritt eine Dummy-Datei auf alle E/A-Knoten geschrieben um die Ausschlusskandidaten zu ermitteln. Danach werden im zweiten Schritt die Anwendungsdaten geschrieben. Die gesamte Funktionalität wurde von Son et al. in PnetCDF implementiert ohne die Schnittstelle zu verändern.

4 Grundlagen

In diesem Kapitel wird auf die technischen Grundlagen und insbesondere das analysierte Ökosystemmodell ECOHAM5 eingegangen.

4.1 ECOHAM5

4.1.1 Das Ökosystemmodell ECOHAM

Für die Untersuchung wurde das an der Universität Hamburg entwickelte biogeochemische Ökosystem-Modell ECOHAM5 (Ecosystem Model Hamburg, Version 5) verwendet. Das Modell wird genutzt um Fragen des Kohlenstoffflusses in der Nordsee im Rahmen des Klimawandels (Lorkowski et al., [LPMK12]; Pätsch und Kühn, [PK08]) sowie Fragen zur Auswirkung unterschiedlicher Belastung des Ökosystems Nordsee durch Nährstoffeinträge von Stickstoff und Phosphor (Lenhart et al. [LMBB$^+$10, LDG$^+$13]) zu untersuchen. Innerhalb von ECOHAM wird die Ökologie der Nordsee für die unteren trophischen Stufen bis hin zum tierischen Plankton (z.B. kleine Krebstiere) abgebildet. Dazu wird die Nordsee in ein dreidimensionales Gitter unterteilt und für jede Gitterzelle werden für eine Reihe von Zustandsvariablen numerische Differenzialgleichungen gelöst. Die verschiedenen Zustandsvariablen repräsentieren die sogenannten funktionellen Gruppen im Ökosystem und wurden so gewählt, dass ihr Zusammenspiel die Kreisläufe bestimmter wichtiger Stoffgruppen abbildet. ECOHAM repräsentiert die Zyklen von Kohlenstoff (C), Stickstoff (N), Phosphor (P), Silikat (Si) und Sauerstoff (O2) mittels der berechneten Zustandsvariablen und den zwischen den Zustandsvariablen agierenden Prozessen. Die Zustandsvariablen umfassen vier Nährstoffe, je zwei Gruppen pflanzlichen (Algen bzw. Phytoplankton) und tierischen Planktons (Zooplankton), langsam und schnell sinkenden Detritus, labiles und semi-labiles gelöstes organisches Material (DOM), Bakterien, gelösten anorganischen Kohlenstoff (DIC), Gesamtalkalinität (TA) und Sauerstoff. Das Phytoplankton sorgt je nach Verfügbarkeit von Nährstoffen und Licht für den Aufbau organischer Substanz durch Phytosynthese. Dieses Phytoplankton wird dann vom Zooplankton gefressen, das dann als Nahrung für Tiere in den höheren Stufen des Nahrungsnetzes dient. Durch das Absterben organischen Materials (Phytoplankton, Zooplankton) entsteht Detritus, der zu Boden sinkt und dort von Bakterien abgebaut wird. Hierbei werden auf der einen Seite Nährstoffe freigesetzt und somit der Nährstoffkreislauf geschlossen, andererseits verbrauchen die Bakterien beim Abbau Sauerstoff, was bei der Unterschreitung einer kritischen Konzentration zu Schäden am Ökosystem führen kann, bis hin zum Absterben bodennah lebender Spezies (Topcu et al,. [TBC09]). Ein weiterer

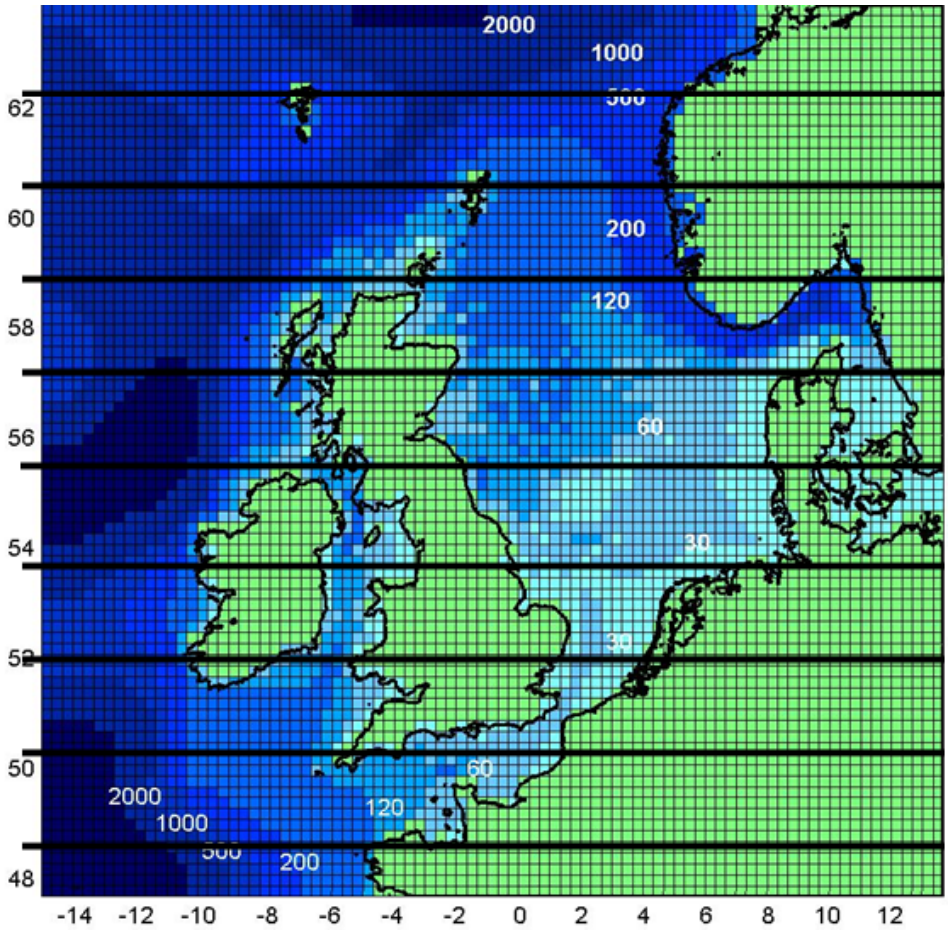

Abbildung 4.1: Horizontales Gitter und Bodentopographie des ECOHAM-Modellgebietes

Fokus der Anwendung von ECOHAMN liegt in die Untersuchung dieser sogenannten Sauerstoffdefizit-Problematik (Große et al., [GGK+16])

4.1.1.1 Modell-Setup und Antriebsdaten

Das Modellgebiet des ECOHAM-Gitters umfasst den Nordwesteuropäischen Kontinentalschelf (NECS) und Teile des angrenzenden Nordostatlantiks (Abbildung 4.1). Das Modell beinhaltet das Gebiet von 15.25 ° West bis 14.083° Ost und von 47.583° bis 63.983° Nord. Die horizontale Auflösung ist 1/5° bei 82 Gitterpunkten in der Breite und 1/3° bei 88 Gitterpunkten in der Länge. Dies entspricht in etwa einer Auflösung von 20x20 km pro Gitterzelle. Die vertikale Dimension umfasst 31 Schichten mit einer Auflösung von 5 m in den oberen Schichten (ausgenommen der Oberflächenschicht mit 10 m) bis 50 m Tiefe. Darunter nimmt die Schichtdicke kontinuierlich zu bis zu einem Maximalwert von 1000 m für die tiefste Schicht mit einer Bodentiefe von 4000 m. ECOHAM kann derzeit Simulationen für den Zeitraum von 1977 bis 2014 unter Verwendung realistischer Antriebsdaten liefern. Das Modell läuft im sogenannten Offline-Modus mit

einem Zeitschritt von 30 Minuten. Die Offline-Kopplung bedeutet, dass das Modell während einer Simulation nicht im direkten Austausch mit einem Strömungsmodell steht, sondern bereits vorliegende von einem Strömungsmodell erzeugte hydrographische und hydrodynamische Felder als Antrieb einliest. Dieses Einlesen erfolgt auf einem Zeitschritt von einem Tag. Die meteorologischen Antriebe mit 6-stündigen Informationen über Lufttemperatur, Bewölkung, relative Luftfeuchtigkeit, Windgeschwindigkeit und -richtung werden dem NCEP/NCAR-Reanalysedatensatz entnommen und auf das Modellgitter interpoliert. Tägliche Frischwasserabflussdaten sowie Nährstofffrachten für 249 Flüsse werden von Cefas (Centre for Environment, Fisheries and Aquaculture Science) bereitgestellt. Die Daten für die atmosphärische Stickstoffdeposition repräsentieren die mittlere Depositionsrate des jeweiligen Jahres.

4.1.2 Implementierung

ECOHAM5 ist ein in Fortran geschriebenes Ökosystemmodell. Es wird mit MPI-2 parallelisiert. Das Modell berechnet die Ökologie der Nordsee ab dem Jahr 1977. Dafür werden mehr als 300 1D, 2D und 3D Variablen berechnet, welche zeitlich und räumlich aufgelöst und mit NetCDF in eine NetCDF-Datei gespeichert werden. Das Modell ist in Fortran90 geschriebenen und kann auf verschiedenen Architekturen genutzt werden. In der Vergangenheit wurde es auf einem IBM System genutzt, den „Blizzard“ des DKRZ (Deutsches Klimarechenzentrum). Nach der Ausmusterung des Blizzards wird es zur Zeit auf dem Bull/Intel-System „Mistral“ und auf einem Intel-Forschungscluster genutzt. Es wird von verschiedenen Institutionen eingesetzt und weiterentwickelt. Der Ablauf des Programms kann grob in drei Phasen unterteilt werden, Start, Simulation, Abschluss. Beim Start des Modells wird es aus den Eingabedaten initialisiert. Der Schritt Simulation kann in zwei Phasen unterteilt werden, die Berechnungs- und die Ausgabephase. Beim Abschluss wird das Modell gestoppt und Daten für eine möglicherweise folgende Simulation werden gespeichert.

ECOHAM5 wurde im Zuge dieser Arbeit auf parallele NetCDF-4 E/A umgestellt, bisher wurde produktiv nur serielle E/A eingesetzt.

4.1.3 Struktur

Die Struktur von ECOHAM5 stellt sich folgendermaßen da:

- Compiler-Script `CompileJob-cluster.sh`
  - Stellt statische Variablen für das Programm ein und kompiliert das Programm mit dem eingestellten Compiler.
  - Bereitet die Ausführung des Programms vor
  - Führt das Programm aus
- Ordner `Input`

  - Enthält Templates welche die zu speichernden Daten konfigurieren: `eco_set.template.nml`
  - Template, welches die biologischen Vorgänge im Modell konfiguriert: `eco_bio.nml`
  - enthält die Warmstart-Dateien um das Modell mit den Daten des Vorjahres zu starten. `warmstart_NWCS20D.in` und `warmstart_NWCS20D_D093.in`

- Ordner `script`
  - Enthält das Laufzeitscript, das die Ausführung des Modells steuert: `RunJobcluster.sh`

- Ordner `src`
  - Enthält den Quellcode des Modells
  - Enthält den Make-Config Ordner mit Konfigurationsdateien für verschiedene Compiler, z.B. den MPI-Compiler mpif90 in `mpif90.config`

- Ordner `extras`
  - Forcing Dateien
  - Grid Definitionen

ECOHAM5 besteht aus den folgenden Fortran-Modulen:

- Fachliche Module
  - `eco_biogeo`
  - `eco_boundaries`
  - `eco_chemie`
  - `eco_flux`
  - `eco_meteo`
  - `eco_par`
  - `eco_rivers`
  - `eco_sediment`
  - `eco_silt`
  - `hydro`

- Grid Module
  - `grid_cskc`
  - `grid_NS03A`
  - `grid_NS20C`

  - grid_NWCS20C
  - grid_NWCS20D
- Hilfsmodule
  - eco_common
  - eco_grid
  - ecoham5
  - eco_init
  - eco_main
  - eco_mpi_parallel
  - eco_restoring
  - eco_sections
  - eco_var
  - utils
- E/A Module
  - eco_ncdfout
  - eco_output_1D
  - eco_output_3D
  - eco_output
  - eco_output_var

Templates ECOHAM5 wird über verschiedene Templates konfiguriert. Für die E/A relevant ist insbesondere das Template `eco_set.template.nml` welches, unter anderem, die Variablen steuert, die von ECOHAM5 gespeichert werden sollen. Weiterhin steuert dieses Template, welche Monate von dem Modell berechnet werden. In dem Auszug 4.1 sind die entsprechenden Variablen aufgeführt.

```
1 iy1                =      YYYY           ! start year
2 im1                =        01           ! start month
3 id1                =        01           ! start day
4 ih1                =        00           ! start hour
5 iy2                =      YYYY           ! ending year
6 im2                =        06           ! ending month
7 id2                =        30           ! ending day
8 ih2                =        24           ! ending hour
```

Listing 4.1: Auszug `eco_set.template.nml`

Scripts Von ECHOHAM5 werden zwei Scipte benutzt, die für diese Arbeit relevant sind: `RunJob-cluster.sh` und `CompileJob-cluster.sh`. Für die E/A-Analyse von ECOHAM5 wurde dabei im `CompileJob-cluster.sh` Script der Compiler auf den Score-P Compiler `scorep` umgestellt, wenn Abläufe mit Tracing durchgeführt wurden. Weiterhin wurden im `RunJob-cluster.sh` Script Umgebungsvariablen für Score-P definiert, die dazu führen, dass Score-P weniger oft seinen Tracingpuffer in das Dateisystem schreiben muss. Weiterhin wurden die Scripte auf die neue Modul-basierte Arbeitsweise umgestellt. Durch die Änderung konnten aktuelle Versionen von NetCDF, HDF5 und MPI genutzt werden.

Im `RunJob-cluster.sh` (vergl. Auszug 4.2) wird auch definiert, welche Jahre berechnet werden sollen. Die initiale Warmstart-Datei liegt für das Jahr 1977 vor. Sollen für ein späteres Jahr Berechnungen durchgeführt werden, so muss das Modell einmal bis zu dem Jahr laufen, damit eine entsprechende Warmstart-Datei vorhanden ist. Für jedes definierte Jahr wird ECOHAM5 einmal ausgeführt. `RunJob-cluster.sh` steuert die Ausführung. Das Modell beendet sich nach Ablauf eines Jahres und wird vom Script dann für das nächste Jahr erneut mit der Warmstartdatei des Vorjahres gestartet.

```
1 set yearStart   = 1977
2 set yearEnd     = 1977
```

Listing 4.2: Auszug `RunJob-cluster.sh`

Compiler-Konfiguration Das `CompileJob-cluster.sh` Script benutzt die Compilerkonfigurationen im Ordner `src/make-config` um daraus die Optionen für den eingestellten Compiler auszulesen. Außerdem werden hier Flags definiert, die festlegen, welche Fortran-Module geladen und dass NetCDF-Daten in Double-Precision geschrieben werden.

```
1 if ( ${mpi_job} >= 1 ) then
2   set FORTRAN_COMPILER = scorep
3   [...]
4 else
5   set FORTRAN_COMPILER = gfortran
6   [...]
7 endif
```

Listing 4.3: Auszug `CompileJob-cluster.sh`

In diesem Fall werden aus der Datei `scorep.config` die Einstellungen geladen.

```
1 FC = scorep mpif90
2
3 ### default options:
4 FFLAGS =-g -p -O3 -xf95-cpp-input -fdefault-real-8
    ↪ -fsign-zero -fno-f2c
```

Listing 4.4: Auszug `scorep.config`

Wenn ein neuer Compiler für die Anwendung benutzt werden soll, so muss erst eine Konfiguration für diesen im Ordner `Make-Config` angelegt werden und Variable `FORTRAN_COMPILER` im `CompileJob-cluster.sh` neu gesetzt werden.

4.1.4 Bibliotheken

ECOHAM5 setzt verschieden Bibliotheken ein:

- MPI-2
- HDF5
- NetCDF

MPI MPI wird für die Kommunikation zwischen den einzelnen Rechenknoten verwendet. Außerdem wird es auch für die Parallelisierung auf einem Rechenknoten eingesetzt, da ECOHAM5 einen Prozess pro CPU-Core ausführt. Dabei kommt MPIs Fähigkeit zum Tragen, effizient innerhalb eines Knotens über geteilten Speicher zu kommunizieren. Außerdem wird MPI-IO als Bibliothek/Middleware von HDF5 zum Speichern in das Dateisystem genutzt. Als konkrete Implementierung wird MPICH in der Version 3.2 genutzt. Siehe auch 2.1 und 2.1.1.

HDF5 HDF5 wird als Bibliothek/Middleware zum Speichern der NetCDF-4 Daten genutzt. HDF5 wird in der Version 1.10.0-patch1 genutzt. Siehe auch Abschnitt 2.2.

NetCDF NetCDF-4 wird direkt von ECOHAM5 genutzt um die Ausgabedaten zu speichern. Vor der Parallelisierung wurde NetCDF im Classic-Modus genutzt. Nach der Parallelisierung und für den Vergleich der zwei Versionen wird NetCDF mit dem NetCDF-4 Dateiformat genutzt. Für die parallele E/A werden die parallelen E/A Fähigkeiten von NetCDF-4, die auf HDF5 basieren, genutzt. PnetCDF wird nicht genutzt. NetCDF wird in der Version 4.4.1 und der Fortran Wrapper um NetCDF wird in Version 4.4.4 eingesetzt. Siehe auch 2.3.

Score-P Score-P wird zum Tracing und Instrumentieren von ECOHAM5 genutzt. Score-P wird in der Version 2.0.2 eingesetzt.

4.1.5 Ablauf

Kompilieren & Ausführen Um ECOHAM5 zu kompilieren muss das `CompileJob-cluster.sh` Script aufgerufen werden. Das Script hat zwei Argumente `RunID` und den Ausführungsmodus. Bei `RunID` kann eine beliebige Text-ID angegeben werden. Beim Argument Ausführungsmodus kann `0, 1` oder `2` als Wert angegeben werden:

- `0`

  - Nur Kompilieren

- `1`
  - Kompilieren und Ausführung vorbereiten (kopiert die Daten auf die Rechenknoten)

- `2`
  - Kompilieren und Ausführen

`CompileJob-cluster.sh` ruft den konfigurierten Kompilier auf und baut das Modell. Die ausführbare Datei liegt dabei im `wrk` Ordner. Das kompilierte Programm wird für die Ausführung auf die Rechenknoten kopiert. Auf diesen wird dann das `RunJob-cluster.sh` Script gestartet, das dann den weiteren Ablauf steuert. Dabei überwacht das Script welche Jahre schon berechnet wurden und welche noch zu berechnen sind (vergl. Script 4.2). Das `RunJob-cluster.sh` startet das Modell für jedes Jahr mit der Warmstart-Datei des Vorjahres. Auf jedem Rechenknoten werden je nach Konfiguration Prozesse gestartet.

Berechnung & E/A Das Modell von ECOHAM5 berechnet die Ökologie basierend auf diskreten Zeitschritten. Während der Berechnung wird die Kommunikation zwischen den Rechenknoten über MPI durchgeführt, siehe auch Abschnitt 4.1.4. ECOHAM5 erzeugt eine NetCDF Datei als Ergebnis der Berechnung. Welche Daten gespeichert werden, wird über das `eco_set_template.nml` gesteuert (vergl. Abschnitt 4.1.3). Dort können mehr als 300 Variablen oder Variablengruppen definiert werden. Diese werden dann in die Ausgabedatei geschrieben.

Von ECOHAM5 wird beim Start der Simulation ein Domain-Decomposition/Splitting vorgenommen. Dabei wird das Simulationsgebiet auf die Rechenknoten verteilt. Im Falle von ECOHAM5 wird das Berechnungsgebiet in Breitengradrichtung unterteilt. Dafür werden 82 Gitterreihen auf die Prozesse aufgeteilt, diese Aufteilung ist in Abbildung 4.1 exemplarisch für 10 Prozesse gezeigt. Jeder Prozesse bekommt 10% des Gebiets zugeteilt. Nicht jeder Abschnitt enthält gleich viele nasse Punkte. Nasse Punkte sind Schnittpunkte im Gitter, die im Wasser sind. Nur auf diesen Punkten werden Berechnungen durchgeführt. Ein Beispiel für das Ungleichgewicht der Verteilung ist, dass der unterste Abschnitt in Abbildung 4.1 nur relativ wenige nasse Punkte enthält. Die Anzahl der nassen Punkte pro Prozess hat Auswirkungen auf die Berechnungsdauer und Ausgabegröße dieses Prozesses. Insgesamt werden 4455 Wassersäulen und 83558 nasse Punkte berechnet und ausgegeben. Nach dem Aufteilen des Gebiets werden die Warmstartdateien eingelesen (siehe unten) und ein Restoring wird durchgeführt. Dabei werden die Werte für jeden nassen Punkt wiederhergestellt bzw. initial gesetzt.

Über das Templat `eco_set_template.nml` kann eingestellt werden, welche Tage und Monate berechnet werden sollen (vergl. Abschnitt 4.1.3). Im Script `RunJob-cluster.sh` können die zu berechnenden Jahre definiert werden (vergl. Abschnitt 4.1.3). ECOHAM5 wird dann von dem Script jahresweise ausgeführt. Nach jedem Jahr wird eine Warmstartdatei geschrieben und die Prozesse beendet. Für das nächste Jahr werden neue Prozesse

auf allen beteiligten Knoten vom `RunJob-cluster.sh` gestartet. Diese lesen dann die Warmstartdatei ein und berechnen das nächste Jahr. Das Modell erlaubt es auch die Monate einzustellen, die berechnet werden. Dies ist sinnvoll, wenn weniger als ein Jahr berechnet werden soll.

Die Ausgabedatei bildet einen kompletten 3D-Quader des Berechnungsgebiets von ECOHAM5. Die Produkte der biogeochemischen Simulationen mit ECOHAM werden als tägliche Werte, sowohl an kumulierten Prozessen als auch als Momentaufnahme der Zustandsvariablen, für das gesamte Modellgitter und den gesamten Simulationszeitraum gespeichert.

4.1.6 Initiale E/A Methodik

ECOHAM5 hat drei E/A Phasen, die simultan zu den Programmphasen Start, Simulation und Abschluss stattfinden. Beim Start werden Daten aus der Warmstartdatei eingelesen und damit das Modell initialisiert. In der Simulationsphase werden die Ergebnisse in eine NetCDF Datei gespeichert. Beim Abschluss eines Ablaufs werden die Daten für einen Warmstart in die Warmstartdatei geschrieben. Den Hauptanteil der E/A Operationen wird beim Speichern der Simulationsergebnisse ausgeführt. In der Start- und der Abschlussphase werden nur geringe Mengen, jeweils etwa 71 MiB, an Daten gelesen bzw. geschrieben. Wie bei Abschnitt 5.2.2 zu sehen, ist die Differenz zwischen Gesamtzeit und CPU-Zeit etwa 10 bis 20 Sekunden. Die CPU-Zeit ist die Zeit, die in der Simulationsphase aufgewendet wird. Die Warmstartdateien sind im Verhältnis zu der NetCDF Ausgabedatei der Simulationsphase auch in der Größe klein. Wie erwähnt, ist die Warmstartdatei etwa 71 MiB groß. Dazu kommen die Forceing-Daten, die für 1977 insgesamt etwa 1,2 GiB groß sind. Da bei einem Referenzlauf (vergl. Abschnitt 5.1) 6 Monate berechnet werden und dabei etwa 97 GiB in die Ausgabedatei geschrieben werden, sind die Warmstart- und Forcing-Daten im Verhältnis klein. Aufgrund dieses geringen Einflusses auf die Gesamtzeit und die geringe Dateigrößen wird im Folgenden nur auf die E/A der Simulationsphase eingegangen.

Im ursprünglichen ECOHAM5 wurde nur serielle E/A, wie in Abbildung 4.2 dargestellt, genutzt. Das Modell führte auf n-Knoten die Berechnungen durch. Die dabei anfallenden Daten wurden dann von allen Knoten via MPI über das Netzwerk auf den Master-Knoten übertragen, wenn mehr als ein Knoten beteiligt war. Der Master-Knoten nutzte dann die serielle NetCDF/HDF5-POSIX Funktionalität um die Daten, wiederum über das Netzwerk, in das Lustre-Dateisystem zu schreiben. Insgesamt müssen bei dieser Art der E/A fast alle Daten doppelt über das Netzwerk übertragen werden. Eine Übertragung findet statt um die Daten aus den einzelnen Knoten zum Master zu bewegen, eine zweite um die gleichen Daten in das Lustre-Dateisystem zu speichern. Diese Ineffizienz ist in Abbildung 4.2 verdeutlicht, jeder schwarze Pfeil zeigt eine Operation auf, die über das Netzwerk geht

In ECOHAM5 werden alle berechneten Daten in eine NetCDF-Datei gespeichert. Dabei werden für einen simulierten Monat, mit allen Variablen, etwa 16 GiB an Daten erzeugt. Zusätzlich wird noch eine Warmstartdatei eingelesen und eine neue ausgegeben. Diese Dateien sind aber im Vergleich zur Ausgabedatei mit einer Größe von 71 MiB

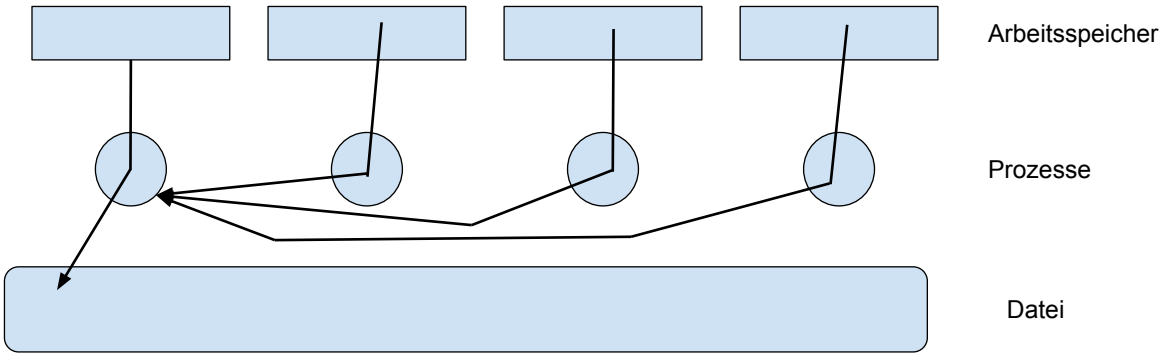

Abbildung 4.2: Sequentielle E/A aus ECOHAM5

klein.

Wie schon oben beschrieben, ist die Ausgabe von ECOHAM5 voll konfigurierbar. Es können verschiedene 1D, 2D und 3D Variablen ausgegeben werden. Davon sind die meisten auch zeitlich aufgelöst. Die meisten 1D Variablen sind Abflussvolumen von wichtigen Flüssen. 2D und 3D Variablen sind über das gesamte Berechnungsgebiet aufgelöst. 2D Variablen sind Variablen die für einen Gitterpunkt spezifiziert sind. 3D Variablen werden zusätzlich noch in die Tiefe aufgelöst.

Diese Daten werden von ECOHAM5 immer am Ende eines simulierten Tages geschrieben. Jeder Tag ist in 48 Zeitschritte unterteilt. Nach den 48 Zeitschritten werden, in der originalen Version, alle Daten auf den Master-Rechenknoten übertragen und dann in die NetCDF Datei geschrieben.

Wie in Abbildung 4.2 zu sehen, werden beim ursprünglichen ECOHAM5 alle E/A-Operationen vom Masterknoten ausgeführt. Dazu werden die Ausgabevariablen in der Ausgabemethode von z.B. dem Modul `eco_output_3D` an den Masterknoten übertragen. Dafür wird MPI eingesetzt. Die Implementierung ist wie im Quellcode 4.5 angegeben. Der Masterknoten ruft MPI_RECV auf, während alle anderen Prozesse mit MPI_SEND alle Daten an den Master übertragen.

```
1  !Aufruf
2     call mpi_parallel_gather(area_master(:,:),1,ierr_mpi)
3
4  !Implementierung
5     if (myID.eq.0) then
6        do r = 1, worldsize-1
7           rowoffset = rowoffset+domainrows(r-1)
8           call MPI_RECV(array(1,1,rowoffset),
               ↪ domainrows(r)*jMax*kDim, &
9                          MPI_REAL8, r, 0, MPI_COMM_ECOHAM,
                              ↪ status, ierr_mpi)
```

```
10        enddo
11     else
12        call MPI_SEND(array(1,1,iStart),
             ↪ domainrows(myID)*jMax*kDim, &
13                      MPI_REAL8, 0, 0, MPI_COMM_ECOHAM,
                           ↪ ierr_mpi)
14     endif
```

Listing 4.5: ECOHAM5 parallel Gather

4.2 Analyse mit Vampir / Score-P

In diesem Abschnitt wird der Einsatz der Analysetools Vampir und Score-P beschrieben. Vampir bzw. Score-P (vergl. Abschnitt 2.5) wurde in dieser Arbeit eingesetzt um das Verhalten des parallelen Programms zu erfassen. (vergl. Abschnitt 2.5) Vampir/Score-P wurde wie in Abschnitt 4.1.3 beschrieben für ECOHAM5 konfiguriert. Es werden die Daten für einen Lauf von einem Monat betrachtet, der sonst wie der Referenzlauf konfiguriert ist (siehe auch Abschnitt 5.1).

4.2.1 Tracing und Analyse

Tracing Die Benutzung von Vampir und Score-P unterteilt sich in zwei Phasen, Tracing und Analyse (vergl. Abschnitt 2.5). Für einen ECOHAM5 Durchlauf von einem Monat fallen etwa 11 GiB komprimierte Daten an. Um diese dann zu verarbeiten und auszuwerten wird der Datensatz komplett in den Arbeitsspeicher geladen, was bei 11 GiB Ausgangsdaten zu etwa 50 GiB Arbeitsspeicherauslastung führt. Zusätzlich zu den Standard Tracing-Daten wurden Tracing-Daten für die POISX-E/A aufgezeichnet. Im seriellen Modus wird von ECOHAM5 und NetCDF-4/HDF5 nur über POSIX-E/A direkt ins Dateisystem geschrieben. Für den parallelen Modus wird NetCDF-4 und HDF5 zusammen mit MPI-IO genutzt, was von Vampir/Score-P standardmäßig aufgezeichnet wird.

Analyse Nachdem die Tracing-Daten, wie in Abschnitt 2.5 beschrieben, gesammelt wurden, folgt die Analyse der Daten. Die Auswertung findet im Vampir Benutzerinterface statt. Es werden die von Vampir bereitgestellten Metriken betrachtet. Dazu zählen Laufzeit in Echtzeit, CPU-Zeit und E/A-Zeit. Weiterhin wird die Filterfunktion auf die verschiedenen E/A-Funktionen und Module angewendet. Dazu zählen die Module `eco_output_3D` und `eco_ncdfout` und die Funktionen init_output und do_output aus diesen Modulen.

Insbesondere der Vergleich der verschiedenen Programmversionen, mit und ohne paralleler E/A, ist für diese Arbeit im Fokus. Für den Vergleich von zwei Tracing-Dateien erlaubt Vampir es Daten im Vergleichsmodus zu laden. Dafür werden zwei Dateien

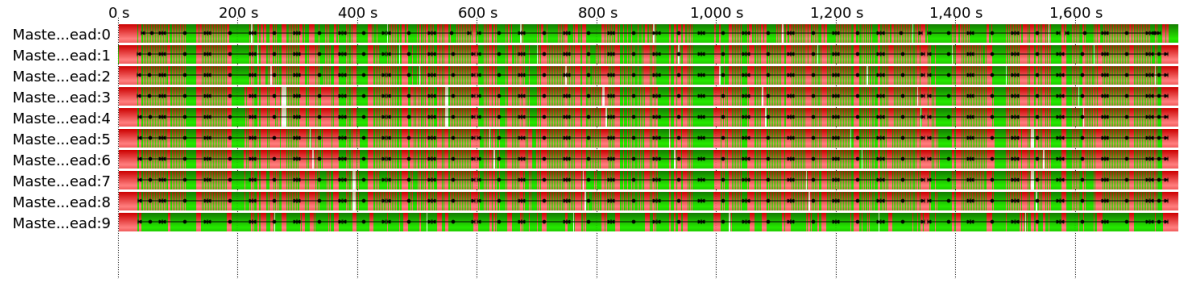

Abbildung 4.3: Übersicht mit Vampir

gleichzeitig geöffnet und ihre Daten zusammen in der Oberfläche angezeigt. Die Fragestellungen für die Analyse sind:

- Gibt es Unterschiede in den Ergebnissen? Liefern die Versionen die gleichen Simulationsergebnisse?
- Was passiert bei der Ausgabe der NetCDF-Datei?
- Welche Unterschiede im Verhalten, insbesondere beim Schreiben der Ausgabe, gibt es zwischen den Versionen?
- Wie stellen sich die Kommunikationsmuster in der Programmversion mit paralleler E/A dar?
- Gibt es Unterschiede bzw. Ungleichgewichte zwischen den einzelnen Knoten?

Die Analyse wird auf einem Knoten mit extra Arbeitsspeicher durchgeführt, magny1. Dieser Knoten hat 110 GiB an Arbeitsspeicher verfügbar. Die Tracing-Daten werden aus dem Lustre-Dateisystem geladen.

Auf den Abbildungen 4.3, 4.5 und 4.6 sind die Details der Analyse dargestellt. In Abbildung 4.3 ist der komplette Ablauf von ECOHAM5 mit serieller E/A zu sehen. In Abbildung 4.4 werden die aufsummierten Zeiten der verschiedenen Funktionsarten dargestellt. Aus diesen beiden Abbildungen wird deutlich, dass fast gleichviel Zeit mit MPI und Anwendungscode verbracht wird. Außerdem sind in der Übersicht (Abbildung 4.3) auf dem Masterknoten grüne Blöcke abgebildet, die nur dort stattfinden. Dies sind die Speichervorgänge, die nur auf dem Masterknoten ausgeführt werden. Im Detail ist dies in den Abbildungen 4.5 und 4.6 zu sehen. Dort sind jeweils im linken Teil der Abbildungen die Pfeile der MPI-Kommunikation zu sehen, mit denen die Daten auf den Masterknoten übertragen werden, wie in Abschnitt 4.1.6 beschrieben. Im Detail sind in Abbildung 4.6 die Aufrufe von `MPI_Send` und `MPI_Recive` dargestellt. Auf diese Kommunikation folgt beim Masterknoten ein durchgehender Teil von Anwendungsaktivität. Dabei werden die Daten im Modul `eco_ncdfout` mit der Funktion `storedata` mit NetCDF gespeichert, wie in Abbildung 4.6 dargestellt.

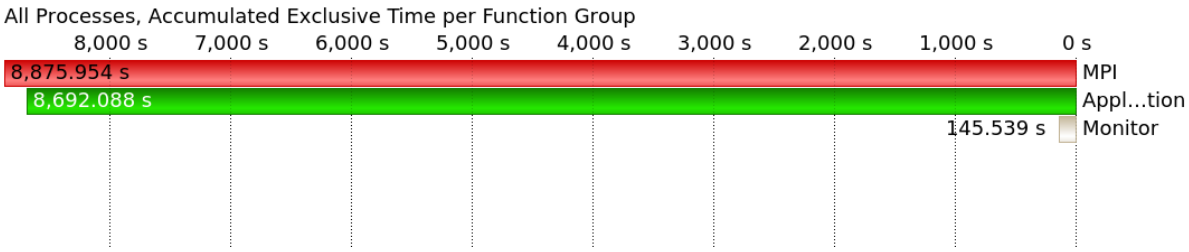

Abbildung 4.4: Aufschlüsselung Zeiten

Abbildung 4.5: Übersicht des E/A Vorgangs

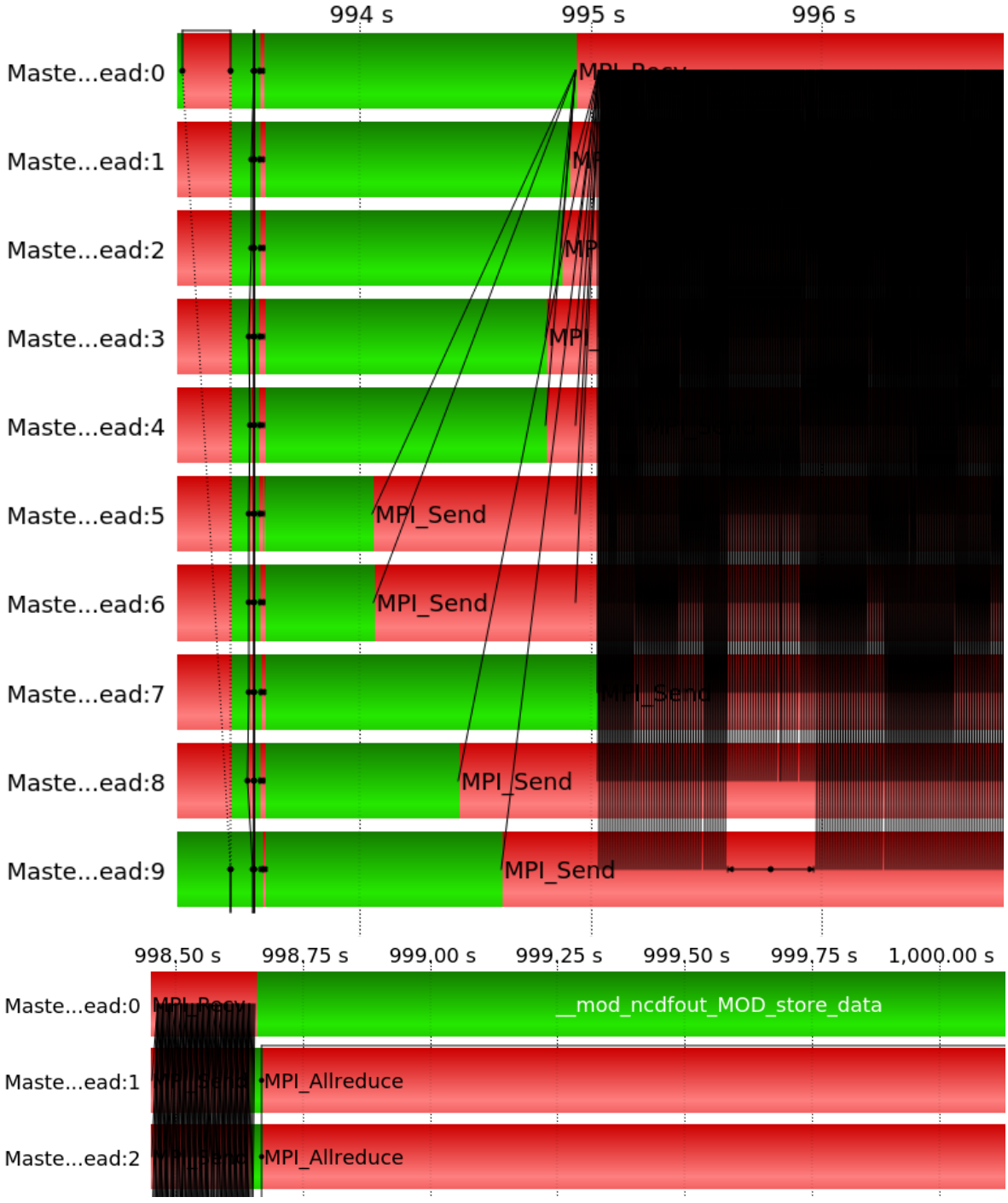

Abbildung 4.6: Details des E/A Vorgangs

5 Evaluation

In diesem Kapitel wird die E/A von ECOHAM5 betrachtet.

5.1 Referenzlauf

Für die Vergleichbarkeit der verschiedenen Abläufe von ECOHAM5 wurde ein Referenzlauf definiert, der im Folgenden beschrieben werden soll. Dieser kann auch in anderen Arbeiten über ECOHAM5 genutzt werden. Die grundlegenden Anforderungen für den Referenzlauf waren:

- Verteilung
  - Wahl zwischen einem und mehreren Rechenknoten
- E/A
  - Genaue Kontrolle, wie viele Variablen ausgegeben werden
  - Single oder Double Precision
  - Abschätzbarkeit der Datenmenge
- Wiederholbarkeit
- Repräsentative Laufzeit
- VampirTrace/Score-P an- und ausschaltbar

5.1.1 Grundlegende Einstellungen

Laufzeit Der Referenzlauf definiert einen Standardzeitraum für die Berechnung. Dies sind die sechs Monate Januar 1977 bis inklusive Juni 1977. Dieser Zeitraum wurde gewählt um eine gute Balance zwischen Laufzeit und einem repräsentativen Querschnitt der Berechnungsdauer abzudecken. Das berechnete halbe Jahr deckt sowohl den ökologisch inaktivieren Winter als auch das ökologisch aktivere Frühjahr ab. (vergl. Quellcode 4.1) Für den Referenzlauf werden daher 181 Tage berechnet.

Verteilung Für die Verteilung kann in `RunJob-cluster.sh`, die Knotenanzahl über SBATCH gesteuert werden, wie im Quellcode 5.1 beschrieben. Für den Referenzlauf werden ein bis zehn Knoten benutzt. Dabei bleibt die Gesamtzahl der Prozesse gleich. Es werden insgesamt 10 Prozesse eingesetzt. Für die Berechnung werden die zehn Hauptrechenknoten des Forschungsclusters genutzt. Die Hardware wird in Abschnitt 1.1 beschrieben.

```
1 #SBATCH --nodes=1
2 #SBATCH --ntasks-per-node=10
```

Listing 5.1: Auszug `RunJob-cluster.sh`

E/A Um vergleichbare und konfigurierbare E/A zu erhalten, werden im Referenzlauf nur 3D-Variablen mit Double-Precision gespeichert. Auf diese wird sich auch konzentriert um den Umfang klein zu halten. Es werden alle Variablen einzeln in `eco_set.template.nml` angegeben und keine Variablengruppen eingesetzt. Alle 3D-Variablen erzeugen dann gleichviel Daten, pro Variable und Zeitschritt 64 Bit also 8 Byte. Mit den 3D-Variablen kann quasi linear zwischen keiner 3D E/A und der maximalen E/A von 320 ausgegebenen Variablen gewählt werden. Die Double-Precision der Ausgabe wird, wie beim Quellcode 5.2 zu sehen, im `makefile` konfiguriert.

```
1 DEFINES   += -DNETCDF_DOUBLE#     # double precision output
    ↪ for netcdf file
```

Listing 5.2: Auszug `makefile`

Für jede Variable, wenn sie zeitlich und räumlich aufgelöst ist, werden an jedem Gitterpunkt für jede definierte Tiefe, ein Datenpunkt mit 8 Byte Nutzdaten gespeichert. Für eine Variable wird bis zu 1,8 MiB[1] gespeichert.

Wiederholbarkeit Der Referenzlauf erzeugt immer gleiche Daten, solange alle Berechnungen auf der gleichen Prozessorarchitektur ausgeführt werden. Berechnungsergebnisse in ECOHAM5 sind abhängig von der prozessorinternen Reihenfolge. Für diese Arbeit werden alle Berechnungen auf Intel x86-64 Systemen durchgeführt. (vergl. 1.1) Das Model nutzt keine Zufallszahlen für die Berechnungen, da es ein numerisches Modell, ist in dem die Differenzialgleichungen immer gleich berechnet werden.

Einschränkungen ECOHAM5 gibt standardmäßig fünf Variablen aus, die nicht konfiguriert werden können. Diese Variablen enthalten grundsätzliche Basisinformationen. Dazu zählen die zwei 2D-Variablen `bathymetry` und `area` und die 3D-Variable `vol0`, welche nicht temporal aufgelöst sind. Außerdem zählen dazu auch die 2D-Variable `dz` und die 3D-Variable `vol`, welche temporal aufgelöst sind. Außerdem werden die Koordinaten-Variablen gespeichert, die das Gitter auf Längen- und Breitengrade abbilden. Diese Variablen können die E/A Skalierbarkeit verzerren. Im Detail bilden die Variablen einen Sockel, unter den die E/A nicht abgesenkt werden kann.

[1] 88*82*31*8 byte = 1789568 byte

5.2 Basisperformance

Die Grundperformance von ECOHAM5 ist durch die Infrastruktur begrenzt. Da die E/A seriell durchgeführt wird, ist die maximale Bandbreite in das Lustre Dateisystem auf 100MiB/s begrenzt (vergl. 1.1), da nur ein Rechenknoten in das Dateisystem schreibt. Weiterhin ist die E/A Leistung eingeschränkt, da vor dem Speichern alle Daten erst zum Master-Rechenknoten übertragen werden müssen. Insbesondere bei der Berechnung mit mehreren Rechenknoten ist dies eine Einschränkung. Die Performance wird erstens über die Ausgaben von ECOHAM5 gemessen. Dazu zählen die Ausgabe von CPU-Zeit und Realzeit, die vergangen sind. Auch die Größe der Ausgabedatei geht in die Messung ein. Zweitens wird die Performance über das Tracingtool Vampir/Score-P gemessen, wie es in Abschnitt 2.5 beschrieben ist. Die Instrumentierung beeinflusst die Gesamtlaufzeit, das Tool versucht diese Einflüsse auf die Messungen zu minimieren und rechnet die Einflüsse aus der Messung herraus.

In diesem Abschnitt soll die Basisperformance von ECOHAM5 festgestellt werden, damit die Version mit paralleler E/A mit diesen Werten verglichen werden kann.

5.2.1 Vorgehen

Für die Ermittlung der Performance wird das folgende Vorgehen gewählt. ECOHAM5 wird in zwei verschiedenen Knotenkonfigurationen gestartet. Bei der ersten Konfiguration wird ein Rechenknoten aus dem Forschungscluster gewählt und auf diesem wird das Programm mit zehn Prozessen (Tasks) gestartet. So werden insgesamt zehn Prozesse von ECOHAM5 ausgeführt. Für die zweite Konfiguration werden zehn gleiche Rechenknoten aus dem Forschungscluster gewählt und auf jedem Rechenknoten wird ein Prozess gestartet. So werden auch in dieser Konfiguration zehn Prozesse von ECOHAM5 ausgeführt.

Als zweite Größe wird die Anzahl der ausgegebenen Variablen von ECOHAM5 verändert. Da im Referenzlauf nur 3D Variablen ausgegebn werden, kann die Anzahl der Variablen über ein Feld in der `eco_set.template.nml` definiert werden, wie in Quellcode 5.3 zu sehen.

```
1 n_str3D_out      =      320          ! number of 3D
    ↪ output-strings (var3D_name) to be read
```

Listing 5.3: Anzahl der Ausgabevariablen definieren

Es werden drei Abläufe des Referenzlaufs mit jeweils verschiedener Anzahl an Variablen durchgeführt. Referenzläufe werden mit keiner Variable (0%), mit 160 Variabeln (50%) und mit allen 320 Variablen (100%) durchgeführt. Die Abläufe werden zusätzlich einmal mit Score-P durchgeführt. Insgesamt werden zwölf verschiedene Abläufe durchgeführt.

Im Quellcode von ECOHAM5 wurden vier Änderungen vorgenommen um das Messen der E/A Zeiten zu ermöglichen. Dafür wurde in den Modulen `eco_output_3D` und `eco_ncdfout` die Laufzeit von jeweils der init und der do_output Subroutinen gemessen.

Diese Zeiten werden beim Beenden von ECOHAM5 für jeden Rechenknoten in das Log geschrieben, wie im Quellcode 5.4 zu sehen.

```
1 [eco_ncdfout.f90]
2 call CPU_Time(ncdf_ts)
3 [...]
4 call CPU_Time(ncdf_te)
5 ncdf_total = ncdf_total + (ncdf_te - ncdf_ts)
6
7 [eco_main.f90]
8 write(log_out, *)  "rank", myID, "I/O Zeit", io_total,
    ↪ "NetCDF Zeit", ncdf_total
9 call write_log_message(log_out)
```

Listing 5.4: Zeitmessung in E/A Code

Cluster Setup Diese Arbeit wurde auf einem Forschungscluster der Universität Hamburg durchgeführt. Der Cluster besteht aus homogenen Knoten:

- 11 Knoten: 1 Login, 10 Compute
- Compute Knoten:
  - 2 Prozessoren (Intel Xeon Westmere 5650 @ 2.67GHz) mit jeweils 6 physischen / 12 logischen Kernen
  - 12 GiB DDR3 / PC1333 Hauptspeicher (System taktet mit 1333 MHz)
  - 2 x Gigabit-Ethernet
  - Lustre Dateisystem
  - Slurm-Jobverwaltung
  - Standardbibliotheken
    * OpenMPI
    * NetCDF-4
    * HDF5

Pro Benutzer können maximal 100 MB/s über das Netzwerk übertragen werden. Diese schließt Dateisystem-E/A über Lustre mit ein.

5.2.2 Performance

Wie oben erwähnt, werden die Gesamtlaufzeit, die CPU-Zeit und die Größe der Ausgabe festgehalten. Zusätzlich wird die Zeit, die ECOHAM5 mit den Modulen `eco_output_3D` und `eco_ncdfout` verbringt, gemessen. Diese Werte werden in Tabelle 5.1 dargestellt.

Lauf | Zeit*	Gesamt	CPU	Dateigröße	eco_-out-put_3D	eco_-ncdfout
1 Knoten á 10 Tasks					
mpif90 0% Variablen	9703 s 161 min	9692 s	934 MiB	42 s	3 s
mpif90 50% Variablen	9969 s 166 min	9955 s	49 GiB	247 s	96 s
mpif90 100% Variablen	10549 s 175 min	10533 s	97 GiB	434 s	188 s
10 Knoten á 1 Task					
mpif90 0% Variablen	6344 s 105 min	6294 s	934 MiB	38 s	1,5 s
mpif90 50% Variablen	6925 s 115 min	6868 s	49 GiB	521 s	84 s
mpif90 100% Variablen	7472 s 124 min	7408 s	97 GiB	1168 s	168 s

*Wenn anwendbar

Tabelle 5.1: Performancedaten ECOHAM5

Da bei ECOHAM5, wie bei Abschnitt 5.1.1 dargestellt, die Ausgabe von bestimmten Variablen nicht abgestellt werden kann, bezeichnet 0% hier die minimal mögliche Anzahl an Variablen. Die Prozentwerte, 0%, 50% und 100%, beziehen sich auf die 320 3D Variablen, die im Referenzlauf definiert sind.

Gesamtzeit Die Gesamtzeit beschreibt die Zeit die insgesamt mit der Ausführung von ECOHAM5 auf dem oder den Rechenknoten verbracht wurde. Die Gesamtzeiten auf einem Knoten liegen nahe beieinander. Sie beginnen für den Programmlauf mit 0% Variablen bei 161 Minuten steigern sich bei 50% der Variablen auf 166 Minuten und enden dann bei 100% der Variablen mit 174 Minuten, wie in Tabelle 5.1 zu sehen. Die Programmzeiten für die Programmläufe auf zehn Rechenknoten variieren deutlich stärker. Außerdem ist die Berechnung insgesamt auf zehn Rechenknoten, unabhängig von der Variablen Konfiguration, etwa 50 Minuten schneller, was einer Steigerung von 30% entspricht. Der Grund für die Beschleunigung liegt an der insgesamt geringeren Auslastung der einzelnen Rechennoten, wodurch beispielsweise die Prozessoren mehr Zeit im Turbo-Modus verbringen können und weniger aufgrund der Wärme gedrosselt werden. Die Verwendeten CPUs haben einen Turbo-Modus mit 15% Steigerung des CPU-Takts (vergl Intel [Int16a]). Für die jeweiligen Konfigurationen sind die zeitlichen Abständen zwischen den Programmläufen, gleich geblieben. So ist der Abstand zwischen den 50% und 100% Programmläufen sowohl bei zehn Rechenknoten als auch bei einem Rechenknoten mit 9 Minuten gleich geblieben. Unter der Annahme, dass diese zusätzliche Zeitspanne mit der Ausgabe der um 48 GiB größeren Datenmenge belegt ist, erreichte das

Programm dabei etwa 88 MiB/s.

CPU-Zeit Die CPU-Zeit ist die Zeit, die mit der Simulation verbracht wurde. Sie unterscheidet sich bei ECOHAM5 mit serieller E/A kaum von der Gesamtzeit.

Dateigrößen Die Dateigröße bezeichnet die Größe der NetCDF Datei die von ECOHAM5 als Ergebnis ausgegeben wird. Die Betrachtung der Dateigrößen zeigt, dass der Ansatz zum Skalieren der Ausgabedaten funktioniert. Auch die Ausgabemenge der nicht vermeidbaren Variablen ist, mit unter 1 GiB, aufgeführt. Rechnet man diesen Sockel aus den Daten heraus, so werden bei dem Lauf mit 50% der Variablen 48 GiB an Daten geschrieben. Bei dem Programmlauf mit 100% der Variablen wird exakt das doppelte mit 96GiB ausgegeben. Die Konstanz der Dateigrößen zwischen den Konfigurationen mit zehn Rechenknoten und mit einem Rechenknoten ist erwartungskonform.

eco_output_3D-Zeit Sie beschreibt die Zeit die in dem Modul `eco_output_3D` vergangen ist, und die Zeit, die insgesamt mit E/A-Aktivitäten verbracht wurde. Dazu zählt auch die Zeit die mit der Übertragung der Ausgabedaten an den Masterknoten verbracht wurde.

eco_ncdfout-Zeit Sie beschreibt die Zeit die in dem Modul `eco_ncdfout` vergangen ist.

5.2.3 Skalierung

In Bezug auf die Performance ist die Skalierung von ECOHAM5 zu beachten. Dabei werden Daten zum E/A Verhalten des Modells ermittelt. Zum ermitteln der Skalierung wird das folgende Vorgehen gewählt. ECOHAM5 wird in verschiedenen Konfigurationen ausgeführt. Alle Konfigurationen nutzen zehn Prozesse. Es werden in allen Konfigurationen 202 3D Variablen ausgegeben. Der Simulationszeitraum und andere Werte sind wie im Referenzlauf (vergl. Abschnitt 5.1) definiert. Für den Test wird die Anzahl der Rechenknoten variiert. Es werden zwischen einem und zehn Knoten genutzt. Für jede Zahl an Rechenknoten wird ECOHAM5 einmal ausgeführt. Bei einem solche Lauf wird eine NetCDF Ausgabedatei mit 62 GiB erzeugt.

Die Prozesse werden für diesen Test mit Slurm auf die Rechenknoten verteilt. Slurm wird als Job-Verwaltungsprogramm auf dem Cluster eingesetzt (siehe auch Abschnitt 5.2.1). Slurm verteilt die Prozesse automatisch auf die Rechenknoten. Slurm unterstützt mehrere Methoden um die Verteilung vorzunehmen. Es wurde die standardmäßige Verteilung eingesetzt, die „cyclic“ Verteilung. Bei dieser Verteilungsart werden die Prozesse reihum auf die Rechenknoten verteilt.

Die Ergebnisse sind in Abbildung 5.1 visuell aufbereitet. Die Daten aus dem Skalierungstest zeigen, dass einerseits die Gesamtlaufzeit sinkt, aber die Laufzeit der E/A steigt. Insbesondere gibt es einen großen Sprung zwischen einem und zwei Knoten. Dabei sinkt die Gesamtzeit um etwa 2500 Sekunden bzw. 40 Minuten. Allerdings steigt

Knotenanzahl	Gesamt-Zeit	eco_output_3D-Zeit (master)	eco_output_3D-Zeit (gemittelt)	eco_ncdfout-Zeit
1	10318 s	328 s	235 s	121 s
2	7770 s	651 s	559 s	105 s
3	7092 s	715 s	582 s	109 s
4	6997 s	735 s	568 s	106 s
5	7031 s	737 s	525 s	111 s
6	7064 s	754 s	544 s	113 s
7	7077 s	743 s	530 s	112 s
8	7090 s	744 s	535 s	111 s
9	7016 s	666 s	455 s	115 s
10	7065 s	701 s	455 s	112 s

Tabelle 5.2: Skalierung von ECOHAM5 mit serieller E/A

die Zeit die im Modul `eco_output_3D` aufgewendet wird um das doppelte. Die Laufzeit des Moduls `eco_output_3D` steigt von etwa 5 Minuten auf über 10 Minuten. Die Zeiten von `eco_output_3D` enthalten die Übertragungszeiten zwischen den einzelnen Prozessen. Die Zeit, die im Modul `eco_ncdfout` aufgewendet wird, sinkt von zwei auf einen Knoten um etwa 13%. Allerdings absolut nur um 16 Sekunden. Die Zeit, die mit dem konkreten Speichern in die NetCDF-Datei verbracht wird, ist relativ konstant bei rund 2 Minuten.

Nach Amdahl ist die maximale Beschleunigung mit Hilfe von Parallelisierung durch den seriellen Anteil des Programms begrenzt, siehe auch Abschnitt 1.1. Bei einem Knoten ist die maximale Beschleunigung von ECOHAM5 auf etwa Faktor 32 beschränkt. Bei 10 Knoten schrumpft dieser Faktor auf 10.

5.2.4 Annahmen über das E/A-Verhalten

ECOHAM5 schreibt die Ausgabedatei nur auf dem Master-Rechenknoten, wie in Abschnitt 4.1.6 beschrieben. Das führt zu theoretischen Ineffizienzen bei der Ausgabe. Die Ausgabe muss zweimal übertragen werden, einmal zum Master-Rechenknoten und dann in das Dateisystem. Dieses Vorgehen ist insbesondere bei der Berechnung auf mehreren Rechenknoten möglicherweise ineffizient. Auf nur einem Rechenknoten wird die Übertragung von MPI im Hauptspeicher des Knotens durchgeführt. Das Kopieren von Daten im Hauptspeicher ist Größenordnungen schneller als das Netzwerk, daher wird in dieser Konfiguration keine Verbesserung erwartet. Eine geringe Steigerung kann theoretisch erreicht werden, wenn die indirekte Barriere, welche durch das Senden der Daten an den Master-Rechenknoten geschaffen wird, im parallelen Fall vermiede werden kann. Bei der Berechnung auf einer größeren Anzahl von Knoten sind theoretisch größere Steigerungen möglich. In dieser Konfiguration ist die Übertragung der Daten auf den Master-Rechenknoten über das Netzwerk eine größere Einschränkung. Daten werden

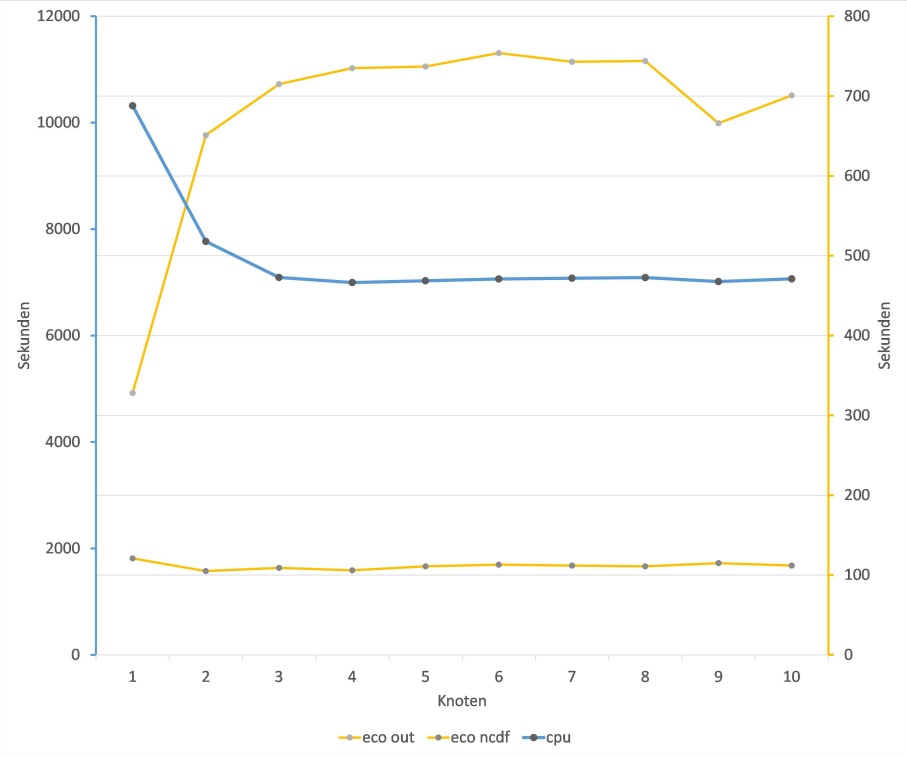

Abbildung 5.1: Skalierung von ECOHAM5 mit serieller E/A

von ECOHAM5 pro simuliertem Tag ausgegeben. Ein Referenzlauf mit allen Variablen von ECOHAM5 erzeugt 97 GiB an Daten, wie in Tabelle 5.1 aufgeschlüsselt. Ein Monat erzeugt etwa 16 GiB an Daten, somit etwa 540 MiB pro Tag. Da jeder Rechenknoten mit 100 MiB/s an das Netzwerk angeschlossen ist, dauert das Speichern, von der Ausgabedatei eines simulierten Monats auf einem Knoten im Idealfall 160 Sekunden. Das Speichern eines Tages dauert etwa 5 bis 6 Sekunden. Da die Berechnung von einem Tag bei 10 Prozessen auf einem Knoten etwa eine Minute dauert, nimmt das Speichern davon unter idealen Bedingungen etwa 8% bis 10% ein. Bei der Berechnung auf mehreren Knoten steigern sich jetzt diese theoretischen Werte auf das Doppelte, aufgrund der doppelten Übertragung. In der für diese Arbeit benutzen parallelen Konfiguration von 10 Rechenknoten mit jeweils einem Task kann durch die Vermeidung der doppelten Übertragung etwa 8% bis 10% der Rechenzeit eingespart werden. Diese Betrachtungen basieren auf der theoretischen Übertragungsgeschwindigkeit / E/A-Bandbreite des Systems. Ob diese Werte erreicht werden können und ob diese Annahmen korrekt sind, wird in Abschnitt 6 genauer betrachtet.

5.3 Parallelisierung ECOHAM5

In diesem Abschnitt wird die Parallelisierung des E/A-Systems des Ökosystemmodells ECOHAM5 dargestellt. Durch die Einführung von paralleler E/A in ECOHAM5 kann

die in Abschnitt 5.2 und in Abschnitt 1.1 beschriebene Limitierung der E/A Bandbreite des Speichersystems auf 100MB/s umgangen werden. Auf dem Forschungscluster ist mit 10 Knoten eine maximale Bandbreite in das Lustre-Dateisystem von 1000 MB/s möglich. Damit könnte die E/A Bandbreite theoretisch um den Faktor zehn gesteigert werden. Um diese theoretischen Verbesserungen nutzen zu können muss das E/A System von ECOHAM5 auf parallele E/A umgestellt werden. Die Hoffnung an die Parallelisierung ist die Senkung der gesamten Laufzeit von ECOHAM5.

5.3.1 Grundlagen und Implementierung

Für die Parallelisierung des E/A-Systems von ECOHAM5 werden die parallelen E/A-Funktionen von NetCDF-4 genutzt. Wie in Abschnitt 2.3 dargestellt, setzt NetCDF für die parallele E/A auf HDF5 und MPI auf. NetCDF nutzt die gleichen Schnittstellen für die parallele E/A wie für die serielle E/A. Dies macht vereinfacht die Implementierung von paralleler E/A. NetCDF ermöglicht es die Ausgabedatei parallel zu öffnen. Dafür muss entweder die Erzeugungsfunktion „Create“ oder die Öffnenfunktion „Open“ mit einem MPI-Kommunikator ergänzt werden. Bei ECOHAM5 wird die Funktion nf90_-create parallelisiert. Dazu wird der MPI-Kommunikator MPI_COMM_NETCDF an die Funktion übergeben (vergl. Quellcode 5.5). Außerdem ist es erforderlich, dass die NetCDF-Datei als NetCDF-4 Datei erzeugt wird und somit auf HDF5 aufbaut, um die parallele E/A nutzen zu können. Dafür muss das Flag NF90_NETCDF4 bei der Erstellung an der Funktion gesetzt werden.

```
1 !seriell
2 call check( nf90_create(ncfile, NF90_NETCDF4, ncid), iret );
    ↪ ierr=ierr+iret
3
4 !parallel
5 mode_flag = ior(NF90_MPIIO, NF90_NETCDF4)
6 call check(nf90_create(ncfile, mode_flag, ncid, comm =
    ↪ MPI_COMM_NETCDF, info = MPI_INFO_NULL), iret);
    ↪ ierr=ierr+iret
```

Listing 5.5: NetCDF nf90_create seriell und parallel

Bei ECOHAM5 ist diese Änderung allein nicht ausreichend. In den Modulen `eco_-output` bzw. `eco_output_3D` wurde sichergestellt, dass nur der Master-Rechenknoten E/A durchführt.

Die Ausgabe von ECOHAM5 läuft über die Module `eco_output, eco_output_1D` bzw. `eco_output_3D`, aus welchen dann das Modul `eco_ncdfout` aufgerufen wird. Dabei ist die Ausgabe in zwei Schritte unterteilt. Im ersten Schritt wird die Ausgabe vorbereitet. Dabei werden in den Modulen die jeweiligen Init-Subroutinen aufgerufen. Diese Beschreibung betrachtet die Module `eco_output_3D` und `eco_ncdfout`. In der Init-Phase wird konfiguriert, welche Variablen ausgegeben werden. Auch wird die NetCDF-Datei erzeugt. Für die Ausgabe werden zwei mehrdimensionale Arrays vorbereitet, full2D und

full3D, welche später die konkreten Daten zu allen ausgegebenen Variablen für einen Zeitschritt enthalten werden. Über MPI werden die Abmessungen der Arrays an alle Knoten übermittelt. Im Modul `eco_ncdfout` wird für jede Variable, die ausgegeben wird, eine Variable in der NetCDF Datei erzeugt, wie beispielhaft im Quellcode 5.6 dargestellt. Auch werden die Dimensionen für Breiten-, Längengrad, Tiefe unter Normal-Null und Zeit definiert. Weiterhin werden in der Init-Subroutine von `eco_ncdfout` die ersten Daten geschrieben. Dazu zählen das konkrete Gebiet, die konkreten Tiefen-Informationen, die Längen- und Breitengrade des Gebiets und die Grid-Informationen.

```
1 call check( nf90_def_dim(ncid, LAT_NAME, NLATS, lat_dimid),
    ↪ iret )
2 call check( nf90_def_var(ncid, LAT_NAME, NF90_REAL,
    ↪ lat_dimid, lat_varid), iret )
```

Listing 5.6: NetCDF anlegen einer Dimension & Variable

Im zweiten Schritt werden die Daten in die NetCDF-Datei geschrieben. In `eco_output_3D` werden in der Subroutine do_output_3D alle Daten des einzelnen Knotens in die Arrays full2D und full3D kopiert. Jeder Knoten überträgt dann seine Kopie von full2D und full3D an den Master-Rechenknoten. Dort werden die Arrays im full2D/3D des Master-Rechenknotens zusammengeführt. Dann wird die Subroutine do_ncdf_out im Modul `eco_ncdfout` auf dem Master-Rechenknoten aufgerufen. In diesem Modul werden die Daten mit nf90_put_var in die NetCDF Datei geschrieben. Vor der Ausgabe müssen die Daten noch anhand des Breitengrads invertiert werden, da das Modell und die Ausgabe von NetcDF für die Visualisierung verschiedene Nullpunkte für Breitengrade haben. Die Ausgabedatei bildet ihren Nullpunkt parallel zu den Geokoordinaten in Richtung Äquator. Der Nullpunkt des Rechengitters liegt Richtung Nordpol. Für diese Invertierung werden die Daten aus den full2D/3D Arrays in temporäre Arrays invertiert überführt. Der Quellcode 5.7 zeigt diese Operation.

```
1 do k = 1, NLVLS
2    do j = 1, NLATS
3       data_out3D(:,j,k) = buffer3D(:,k,NLATS-j+1)
4    enddo
5 enddo
```

Listing 5.7: Rotieren der Ausgabe

Änderungen für parallele E/A Wie oben erwähnt, mussten die Module `eco_output_3D` und `eco_ncdfout` für die parallele E/A angepasst werden, da die E/A Funktionalität nur auf dem Master-Rechenknoten ausgeführt wurde. Auch die Konfigurationsdatei, die die Information, welche der Variablen ausgegeben werden, enthält, wurde nur auf dem Master-Rechenknoten eingelesen und verarbeitet. Diese Einschränkungen wurden entfernt, so dass diese Operationen von allen Prozessen durchgeführt werden. Alle Daten wurden per MPI an den Master-Rechenknoten übertragen. Diese Übertragung ist bei paralleler E/A nicht mehr notwendig und wurde ausgebaut. Das Modul `eco_ncdfout`

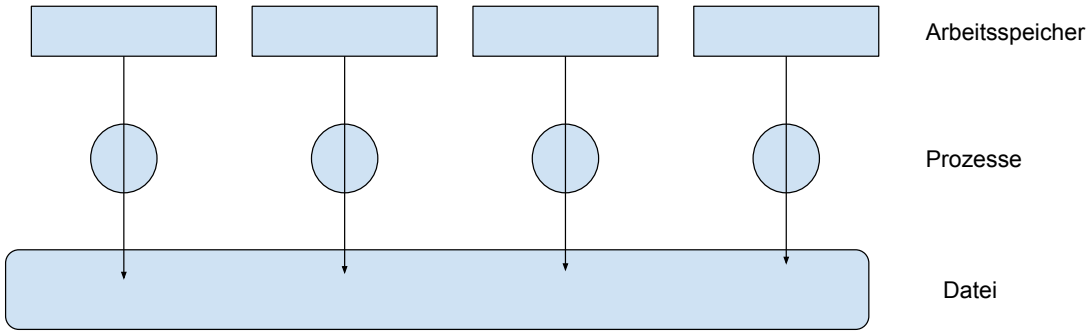

Abbildung 5.2: Parallele E/A

wird dadurch nicht mehr nur auf dem Master, sondern auf allen Prozessen ausgeführt. In diesem Modul wird die NetCDF Datei erstellt und die NetCDF-E/A durchgeführt. Wie oben im Quellcode 5.5 gezeigt, wurde der Aufruf der nf90_create Funktion angepasst. Weiterhin mussten die in Arrays vorgehaltenen Daten für die Ausgabe angepasst werden. Das neue Vorgehen wird in Abbildung 5.2 aufgezeigt. Dabei ist jeder Pfeil die Ausgabe mit NetCDF.

Zuschneiden der Ausgabe-Arrays Das ursprüngliche Vorgehen in ECOHAM5 war, wie oben erläutert, alle Daten für einen Zeitschritt in die full2D und full3D Arrays zusammenzuführen. Diese Sammeloperation ist mit der parallelen Ausgabe nicht mehr notwendig, da jeder Knoten seine Daten selber speichert. Um die Seiteneffekte der Implementierung niedrig zu halten, wurde das Vorgehen für in `eco_output_3D` nicht umfassend geändert. Die Daten des Knotens werden weiterhin in diesen zwei Arrays gesammelt bevor sie an das Modul `eco_ncdfout` übergeben werden. Dort werden die Daten nach der oben erwähnten Invertierung auf das Rechengebiet (local Domain) des Knotens zugeschnitten.

```
1 tmp_arr_2D(:,:) = data_out2D(:,NLATS-iEndD+1:NLATS-iStartD+1)
```

Listing 5.8: Beispiel: Zuschneiden der Ausgabe

Beispielhaft wird das im Quellcode 5.8 gezeigt. Das Array tmp_arr_2D wird später in die NetCDF Datei gespeichert. Das Array data_out2D bildet von der Größe her das komplette Gitter der Simulation ab. Es enthält aber nur Daten eines einzelnen Knotens. Der erste Index des Arrays beschreibt die Gitterlinien nach Längengraden. Die Variable NLATS beschreibt die Gesamtzahl der Gitterlinien nach Breitengraden. iStartD und iEndD enthalten die Start- und Endwerte für das Rechengebiet eines Knotens nach Breitengraden. Wie bei Abschnitt 4.1 erwähnt, wird das Rechengebiet an den Breitengraden entlang auf die Rechenknoten verteilt. Das erste Gebiet an den ersten Knoten usw. Der Zuschnitt von den Ausgabearrays kann nicht von iStartD bis iEndD erfolgen, da die Arrays, wie oben erwähnt, invertiert wurden. Nach der Invertierung ist das erste Gebiet

nun am Ende des Arrays. In den Arrays wie tmp_arr_2D sind nach dem Zuschneiden nur noch die Daten eines Rechenknotens enthalten. Diese Daten können dann in die NetCDF Datei gespeichert werden. Wie in Quellcode 5.9 zu sehen, werden beim Schreiben von Daten in eine Variable die Werte start und count übergeben. Mit start kann gesteuert werden, an welcher Position in der Variable die Daten gespeichert werden. Mit count kann die Menge der zu speichernden Daten begrenzt werden.

```
1 nc_start = (/ dim1, NLATS-iEndD+1, dim1, dim1 /)
2 nc_count = (/ NLONS, iEndD-iStartD+1, NLVLS, dim1 /)
3 [...]
4 call check( nf90_put_var(ncid, id, r4_3d, start=nc_start,
    ↪ count=nc_count), iret )
```

Listing 5.9: NetCDF Daten in Variable speichern

Die meisten Variablen von ECOHAM5 werden im Collective Modus ausgegeben. Im Collective Modus ist der Aufruf von NetCDF-E/A-Operationen mit einer Barrier an diesem Aufruf verbunden. Dadurch rufen alle Rechenknoten die E/A Operation gleichzeitig auf. Die Nutzung des Independet Modus ist bei ECOHAM5 derzeit nicht möglich, da NetCDF das parallele Vergrößern von ungebundenen Dimensionen (unbound dimensions) im Independent Modus nicht unterstützt. Der Modus kann, wie bei Quellcode 5.10 gezeigt, für jede Variable einzeln gesetzt werden.

```
1 call check(nf90_var_par_access(ncid, rec_varid,
    ↪ nf90_collective), iret)
```

Listing 5.10: Setzen des parallelen Zugriffsmodus

Implementierungsprobleme Wie in Abschnitt 1.1 erwähnt nutzt der Forschungscluster standardmäßig OpenMPI als MPI Implementierung. Die auf dem System installierte Version der Bibliothek lief nicht fehlerfrei mit NetCDF zusammen. Die parallelen E/A Operationen von NetCDF konnten keinen MPI Kommunikator von OpenMPI verwenden. Der Aufruf von NetCDF mit einem OpenMPI Kommunikator führt zum Absturz der Anwendung. Als Lösung stellte sich nur der Wechsel der MPI Implementierung auf MPICH dar. Daraufhin mussten auch NetCDF und HDF5 neu für MPICH kompiliert werden. Dieser Wechsel auf eine andere MPI Implementierung machte auch den Wechsel des Tracingtools notwendig. Nur der Vampir Nachfolger Score-P kann fehlerfrei mit der neuesten Version von MPICH operieren. Weiterhin gibt es einen Bug in NetCDF/HDF5, welcher die Ausgaben von mehr als 202 Variablen nach 3 GiB an Daten pro Prozess mit einem Fehler abbricht.

Laufzeitverhalten Das Laufzeitverhalten von ECOHAM5 mit paralleler E/A wurde mit Vampir analysiert (vergl. 4.2). In Abbildung 5.3 ist die Übersicht über den kompletten Ablauf zu sehen. In Abbildung 5.4 werden die aufsummierten Zeiten der verschieden Funktionsarten dargstellt. In diesen beiden Abbildungen zeigt sich, dass mehr Zeit mit

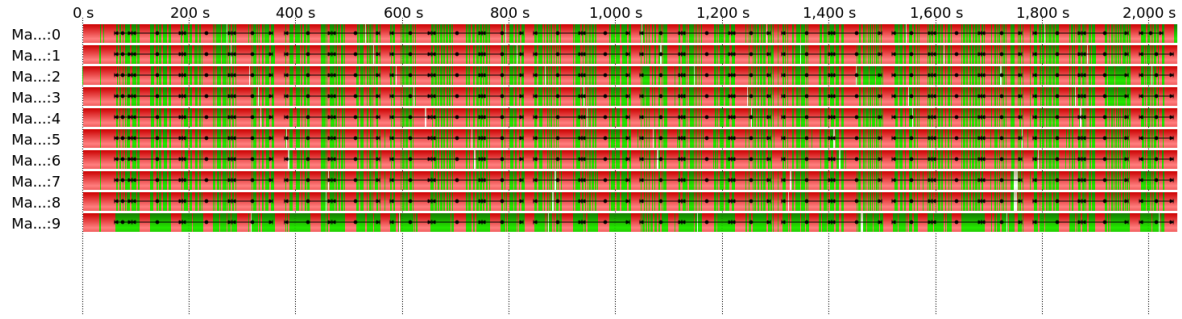

Abbildung 5.3: Übersicht mit Vampir

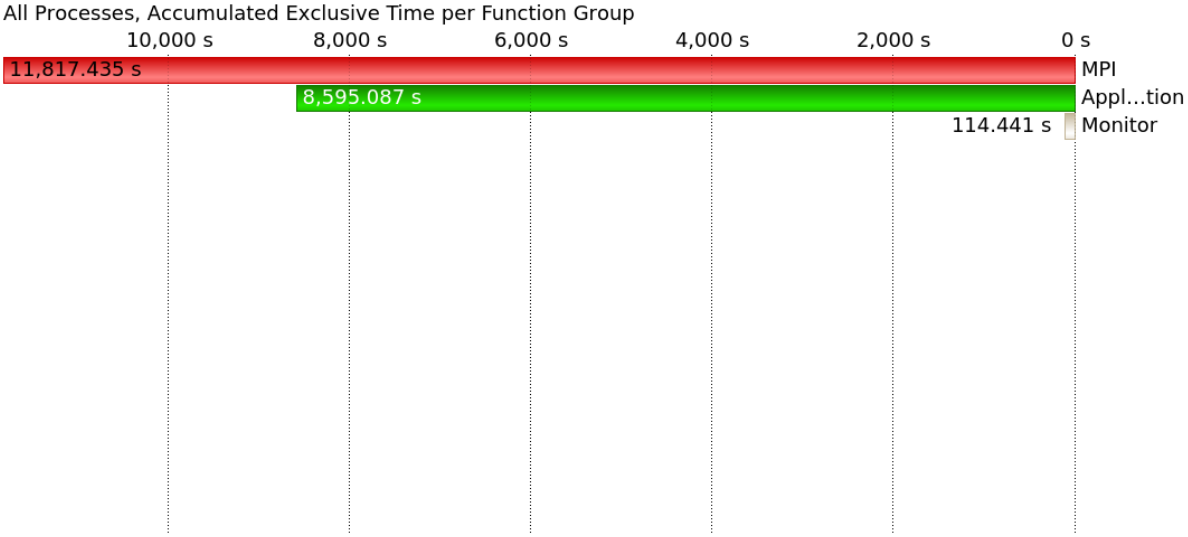

Abbildung 5.4: Aufschlüsselung Zeiten

MPI als mit dem Anwendungscode verbracht wird. Da diese Version MPI-IO via NetCDF einsetzt, werden MPI-IO Funktionen jetzt eingesetzt, wie in den Abbildungen 5.5 und 5.6 abgebildet. Dabei ist Abbildung 5.5 die Übersicht über eine komplette Ausgabephase. Die dünnen grünen Striche sind die Subroutine `eco_ncdfout` und eine weitere Subroutine, die direkt NetCDF aufruft. Wie in Abbildung 5.6 zu sehen, werden von NetCDF bzw. HDF5 drei Funktionen aus MPI und MPI-IO aufgerufen. Diese Methoden sind MPI_Allreduce, MPI_File_write_at_all und MPI_set_view. Die weißen Blöcke in Abbildung 5.5 sind Zeiten, die für Score-P aufgewendet wurden.

Validierung Die Ausgabedateien von ECOHAM5 wurden verglichen. In der Validierung soll gezeigt werden, dass die Ausgabedatei gleich ist. Dafür wurden zwei Tools eingesetzt, ncview [Pie16] und nccmp [Zie15]. Ncview erlaubt es NetCDF-Dateien anzuzeigen, siehe auch Abbildung 5.7. Es können in ncview alle Variabeln einer NetCDF-Datei untersucht und angezeigt werden. Zum Vergleichen von zwei NetCDF-Dateien können diese gleichzeitig in eine Instanz von ncview geladen werden. Operationen können dann gleichzeitig auf den Daten von beiden Dateien durchgeführt werden. Die Daten

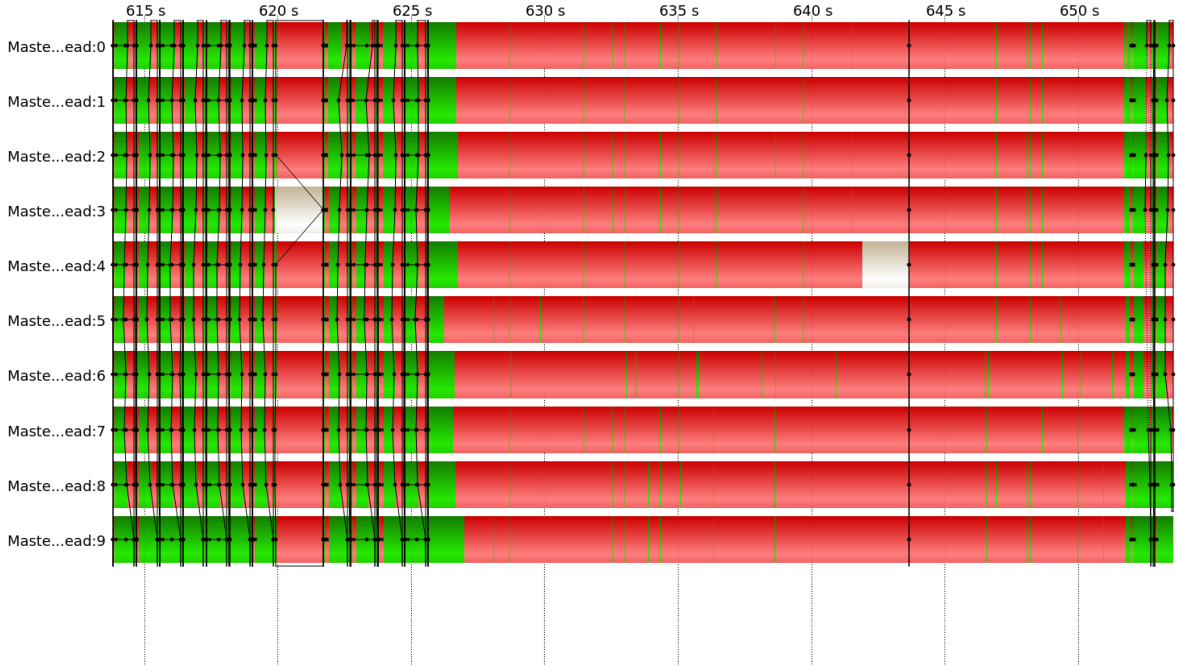

Abbildung 5.5: Übersicht Ausgabephase

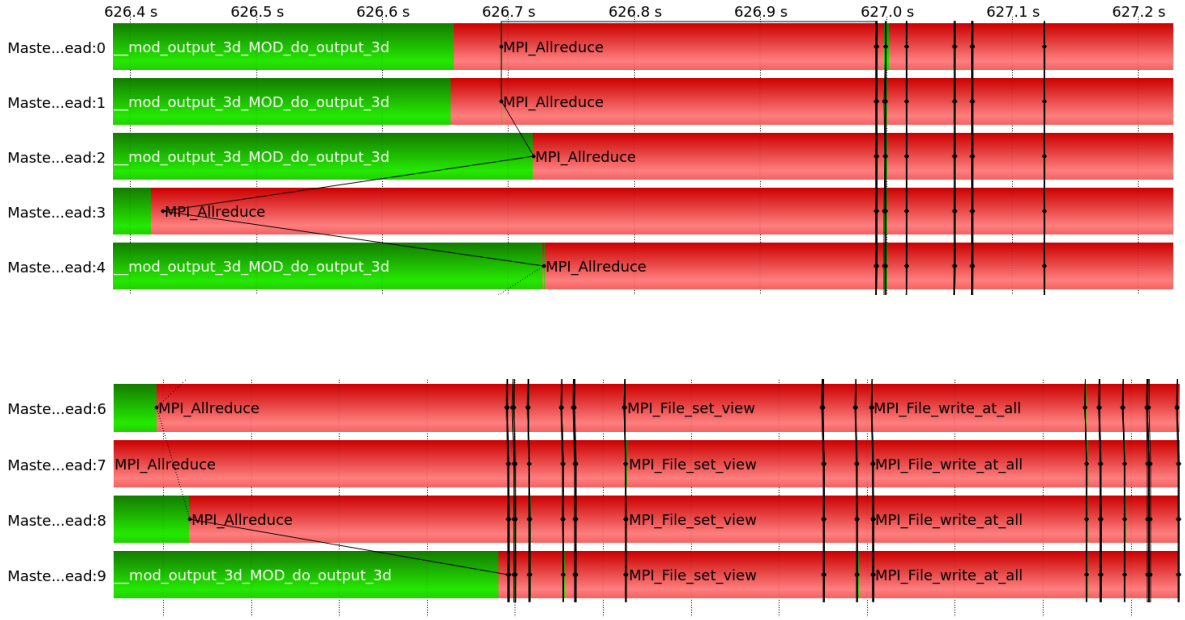

Abbildung 5.6: Details Ausgabephase

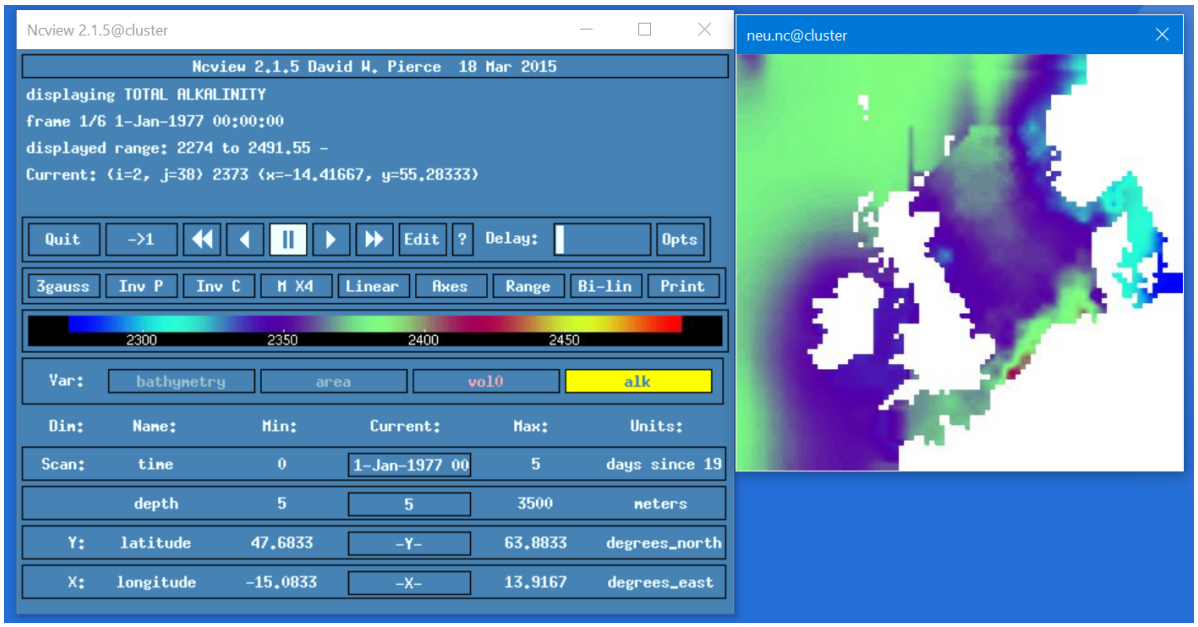

Abbildung 5.7: Screenshot von ncview

werden untereinander im selben Fenster dargestellt. Diese Funktion wurde genutzt um die Ergebnisse von ECOHAM5 mit paralleler E/A mit der Version mit serieller E/A optisch und stichprobenartig zu vergleichen. Die Ausgaben von paralleler und serieller E/A sind visuell nicht zu unterscheiden.

Die Ausgaben wurden mit nccmp verglichen. Dabei wurden sowohl die Metadaten als auch die Daten der NetCDF-Dateien verglichen. Die Ausgabedateien sind laut nccmp identisch, wie in Ausschnitt 5.11 zu sehen.

```
1 % ./nccmp -mds seriell.nc parallel.nc #vergleiche Metadaten
    ↪ und Daten
2 Files "seriell.nc" and "parallel.nc" are identical.
```

Listing 5.11: nccmp Ausgabe

5.3.2 Vorgehen

Für die Version von ECOHAM5 mit paralleler E/A wird das gleiche Vorgehen gewählt wie für die Version mit serieller E/A. Das Vorgehen ist in Abschnitt 5.2.1 dokumentiert. Insgesamt werden auch von ECOHAM5 mit paralleler E/A sechs Abläufe mit verschiedenen Variablenzahlen durchgeführt.

5.3.3 Neue Performance-Daten

In diesem Abschnitt werden die Performancedaten aus Abschnitt 5.2 wieder aufgegriffen und die Messungen mit der Version von ECOHAM5 mit paralleler E/A wiederholt.

Lauf | Zeit*	Gesamt	CPU	Dateigröße	`eco_output_-3D`	`eco_-ncdfout`
1 Knoten á 10 Task					
mpif90 0% Variablen	9392 s 156 min	9380 s	934 MiB	64 s	63 s
mpif90 50% Variablen	9953 s 165 min	9939 s	49 GiB	1133 s	1103 s
mpif90 100% Variablen †	10881 s 181 min	10826 s	97 GiB	2396 s	1942 s
10 Knoten á 1 Task					
mpif90 0% Variablen	6405 s 107 min	6351	934 MiB	97 s	95 s
mpif90 50% Variablen	8122 s 135 min	8057 s	49 GiB	2319 s	2307 s
mpif90 100% Variablen †	9912 s 165 min	9680 s	97 GiB	4675 s	4652 s

* Wenn anwendbar

† Hochrechnung von 60 Tagen

Tabelle 5.3: Performancedaten ECOHAM5 mit paralleler E/A

Die Werte für den Programmlauf mit 100% bzw. 320 Variablen müssen aufgrund eines Fehlers in der Implementierung von 60 Tagen hochgerechnet werden. Diese Werte können daher nicht voll vergleichbar sein.

Gesamtzeit Die Gesamtzeit beschreibt die Zeit, die insgesamt mit der Ausführung von ECOHAM5 auf dem oder den Rechenknoten verbracht wurde. Die Gesamtzeit steigt auf einem Rechenknoten von 0 Variablen auf 160 Variablen um 9 Minuten an. Von 160 auf 320 Variablen steigt die Laufzeit um 16 Minuten. Bei zehn Rechenknoten steigen die Zeiten um 28 Minuten von 0% auf 50% und um 30 Minuten von 50% auf 100% an.

CPU-Zeit Die CPU-Zeit, ist die Zeit, die mit der Simulation verbracht wurde. Sie unterscheidet sich bei ECOHAM5 mit paralleler E/A kaum von der Gesamtzeit.

Dateigröße Die Dateigröße bezeichnet die Größe der NetCDF Datei, die von ECOHAM5 als Ergebnis ausgegeben wird. Die Größe zeigt, wie viele Daten geschrieben wurden. Sie zeigt auch, dass sowohl mit serieller E/A als auch mit paralleler E/A die gleiche Datenmenge geschrieben wird.

Zeit in eco_output_3D Sie beschreibt die Zeit, die in dem Modul `eco_output_3D` angefallen ist und die insgesamt mit E/A verbracht wurde. Bei paralleler E/A wird wenig

Knoten-anzahl	Gesamt-Zeit	eco_output_3D-Zeit	eco_ncdfout-Zeit	eco_output_3D-Zeit (gesamt)	eco_ncdfout-Zeit(gesamt)
1	9980 s	1307 s	1268 s	13073 s	12682 s
2	7879 s	2177 s	2160 s	21772 s	21599 s
3	8388 s	3004 s	2989 s	30040 s	29889 s
4	8512 s	3155 s	3141 s	31554 s	31412 s
5	8959 s	3488 s	3474 s	34884 s	34742 s
6	9071 s	3396 s	3381 s	33956 s	33812 s
7	9141 s	3442 s	3428 s	34423 s	34281 s
8	8538 s	2925 s	2911 s	29248 s	29106 s
9	8354 s	2810 s	2796 s	28103 s	27964 s
10	9198 s	2913 s	2899 s	29125 s	28993 s

Tabelle 5.4: Skalierung von ECOHAM5 mit paralleler E/A

Zeit für `eco_output_3D` aufgewendet, sondern die meiste Zeit wird in `eco_ncdfout` mit der Ausgabe über NetCDF verbracht.

Zeit in eco_ncdfout Sie beschreibt, wie viel Zeit in den Subroutinen von `eco_ncdfout` vergangen ist. Dies ist primär die Zeit, die mit der Ausgabe über NetCDF verbracht wird. In diesem Modul wird bei paralleler E/A der Großteil der E/A Zeit aufgewendet.

5.3.4 Skalierung

Die Skalierung der Version von ECOHAM5 mit paralleler E/A wird so wie in Abschnitt 5.2.3 beschrieben durchgeführt. Tabelle 5.4 zeigt die Gesamtlaufzeit von ECOHAM5 und die Zeiten, welche in den Modulen `eco_output_3D` und `eco_ncdfout` gemessen wurden. Diese werden für die Vergleichbarkeit mit Tabelle 5.2 als Gesamtzeit und als Durchschnittszeit angegeben. Die Messungen der Programmläufe von ECOHAM5 mit paralleler E/A zeigen, das es eine Leistungssteigerung mit mehreren Rechenknoten gibt, wie in Abbildung 5.8 dargestellt. Die Gesamtlaufzeit erreicht ihren niedrigsten Stand bei zwei Rechenknoten. Ab drei Rechenknoten steigt die Laufzeit wieder auf über 8000 Sekunden an. Die Laufzeit steigt für die meisten Knotenzahlen auf etwa 9000 Sekunden, allerdings gibt es bei 8 und 9 Knoten einen Abfall der Gesamtlaufzeit auf 8500 bzw. 8300 Sekunden. Die Zeit, die ECOHAM5 mit E/A Operationen verbringt, steigt allerdings von 1270 Sekunden bei einem Rechenknoten auf 2180 Sekunden für zwei Rechenknoten auf über 3000 Sekunden bei drei oder mehr Rechenknoten. Bei mehr als 8 Rechenknoten fällt die E/A Zeit auf knapp unter 3000 Sekunden. Das Modul `eco_ncdfout` ruft NetCDF auf und zeigt die Zeiten, die direkt mit E/A Operationen verbracht werden. Insgesamt wird wenig Zeit für die Vorbereitung der E/A in `eco_output_3D` aufgewendet, sondern der größte Anteil wird in `eco_ncdfout` aufgewendet. Die Zeiten für `eco_ncdfout` sind, bei mehr als zwei Rechenknoten, relativ stabil bei rund 3000 Sekunden.

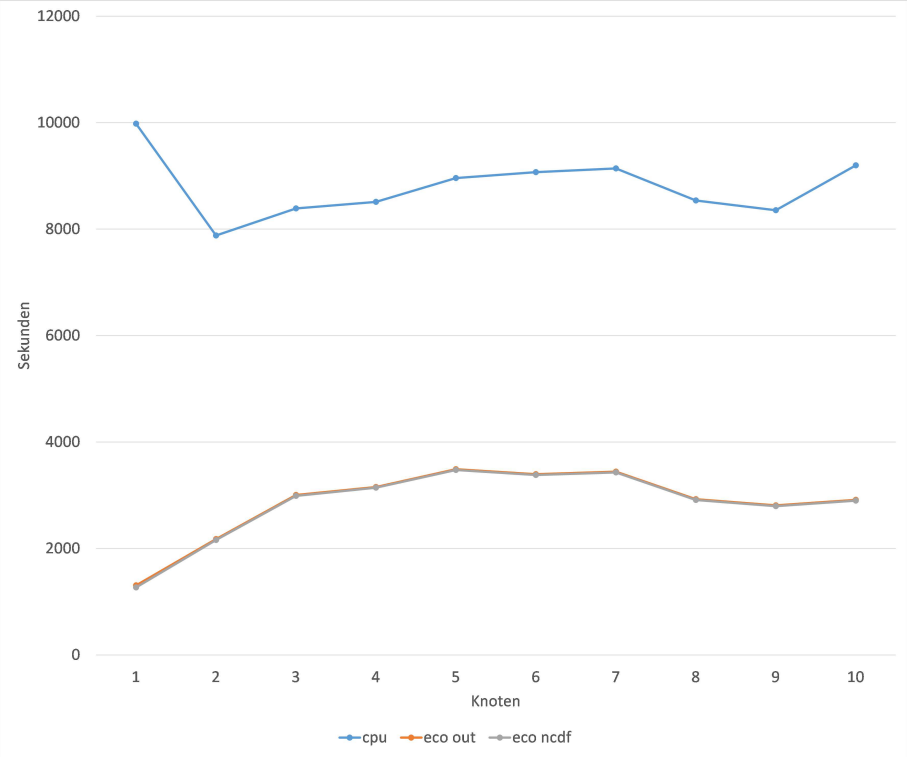

Abbildung 5.8: Skalierung von ECOHAM5 mit paralleler E/A

5.3.5 Probleme mit paralleler E/A

Für diese Arbeit wurde die parallele E/A mit der Schnittstelle von NetCDF implementiert. Diese erlaubt nur eingeschränkte Kontrolle über das Verhalten der darunterliegenden Bibliotheken. Wie schon von Bartz et al. [**Bar14**] gezeigt wurde, hat das Aligment der Ausgabeblöcke mit den Lustre-Stripes einen starken Einfluss auf die E/A Performance von NetCDF im Collective Modus. Dieses Alignment lässt sich nicht über die NetCDF Schnittstelle festlegen, auch wenn diese Fähigkeit in der HDF5-Bibliothek und MPI-IO vorhanden ist. Durch das Fehlen des Alignments wird zusätzlicher Aufwand beim Schreiben der Daten ausgelöst. Das Lustre Dateisystem muss dann auf aufwendigere Operationen und exklusive Sperren zurückgreifen, um die E/A-Vorgänge voneinander zu isolieren.

In ECOHAM5 ist es für die meisten Variablen nicht möglich, die Ausgabe im Independet-Modus vorzunehmen. Dadurch wird jeder Schreibvorgang zu einer Barriere. Aus den Barrieren können Wartezeiten für Prozesse entstehen, die schneller mit der Berechnung fertig waren. Weitere Nachteile bei paralleler E/A in ECOHAM5 können die häufigen und kleinen Speichervorgänge sein. Da in ECOHAM5 nach jedem simuliertem Tag die Daten in die Ausgabe geschrieben werden, sind die Datenmengen, die pro Speichervorgang anfallen, eher gering. Bartz et al. [**Bar14**] zeigen allerdings, dass erst größere Datenmengen zu einer hohen Datenrate führen. Dabei führen sie den Idealwert von 512 MiB Datenvolumen pro Prozess an. Dieser Wert kann von ECOHAM5 nicht erreicht

werden, es werden für jeden simulierten Tag nur etwa 54 MiB an Daten pro Prozess (Bei zehn Prozessen) geschrieben.

6 Analyse

In diesem Kapitel werden die gesammelten Messergebnisse evaluiert und analysiert.

Dieses Kapitel unterteilt sich in drei Abschnitte. In den Abschnitten 6.1 und 6.2 werden die Ergebnisse der beiden Versionen von ECOHAM5 besprochen. Im Abschnitt 6.3 werden die beiden Versionen miteinander verglichen.

6.1 Serielle E/A

Die Version von ECOHAM5 mit serieller E/A ist die originale Version. Die Messdaten sind in Abschnitt 5.2 dargestellt, insbesondere in Tabelle 5.1 und Abbildung 4.2. Erst werden die Laufzeiten und Messwerte der Programmläufe mit unterschiedlicher Variablenzahlen betrachtet. An den Werten ist zu ermitteln, dass ECOHAM5 generell davon profitiert auf mehreren Rechenknoten ausgeführt zu werden, auch wenn dabei die Anzahl der Prozesse nicht steigt. Bei der Berechnung auf zehn Rechenknoten erreicht es in jeder Variablenkonfiguration eine Geschwindigkeitssteigerung von rund 50 Minuten. Das ist eine Steigerung von fast 30%. Als Grund für diese Reduzierung der Laufzeit können verschiedene Faktoren herangezogen werden. Die Steigerung kann zum Teil, wie auch schon in Abschnitt 5.2.2 dargestellt, aus der besseren Ausnutzung des Turbo-Modus der CPUs folgen. Auch die bessere Ausnutzung des Caches des CPUs spielt eine Rolle. Weiterhin tritt insgesamt keine höhere Gesamtlaufzeit durch die Übertragung der Ergebnisse auf den Masterknoten ein. Diese Übertragung wurde in der theoretischen Betrachtung in Abschnitt 5.2.4 als ein Engpass wahrgenommen, der sich auf die Gesamtlaufzeit auswirken würde. Die Steigerungsraten zwischen den jeweiligen Konfigurationen auf einem Rechenknoten und 10 Rechenknoten sind sehr konstant. Bei der Steigerung von 50% der Variablen auf 100% der Variablen steigt sowohl bei einem als auch bei zehn Rechenknoten die Gesamtzeit um zehn Minuten an. Unterschiede gibt es allerdings bei der Steigerung von 0% der Variablen auf 50% der Variablen. Bei diesen Konfigurationen steigt auf nur einem Rechenknoten die Gesamtzeit um fünf Minuten an. Für den Programmablauf mit zehn Rechenknoten gibt es eine Steigerung von zehn Minuten. Dass es diese Steigerung gibt, deutet darauf hin, dass der Engpass doch Auswirkungen hat. Diese Auswirkung tritt bei der Konfiguration mit 100% der Variablen auch im Vergleich mit der 0% Konfiguration auf. Der Einfluss der Datenmenge auf diesen Engpass ist allerdings nicht linear, was die gleichbleibende Steigerung von der 50% Konfiguration auf die 100% Konfiguration zeigt. Die E/A Zeiten zeigen, dass das Netzwerk doch einen Engpass bildet, auch wenn dies in den Gesamtlaufzeiten von der allgemein kürzeren Laufzeit verborgen wird. Dies stellt sich so auch in Abbildung 6.1 dar, da die Laufzeiten

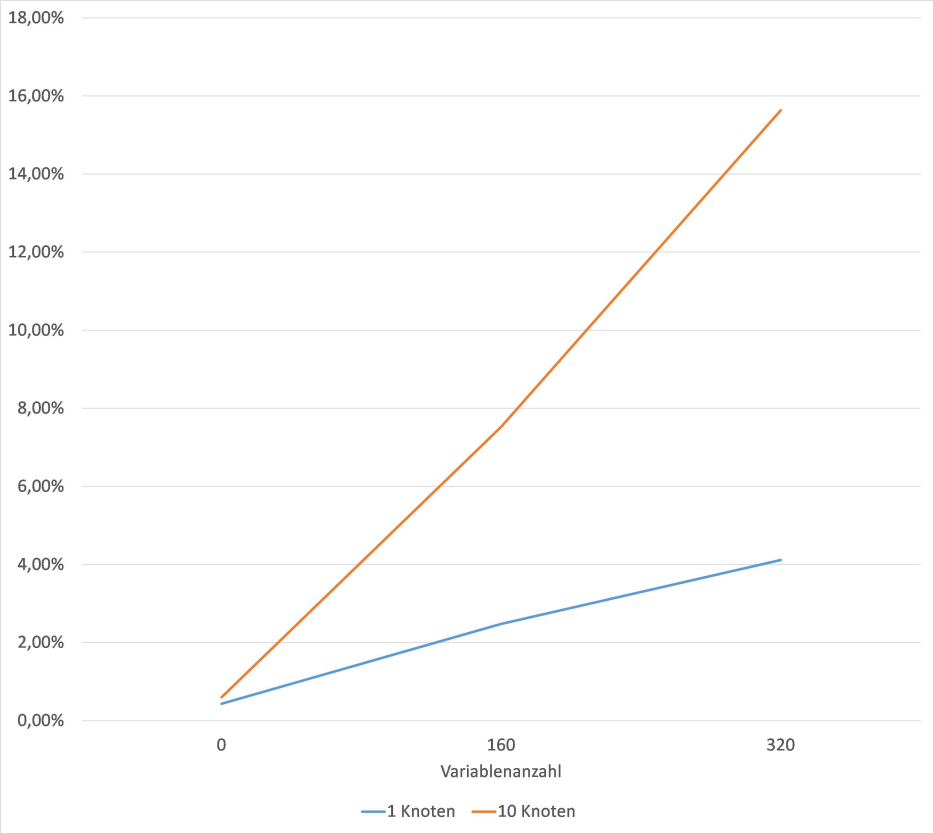

Abbildung 6.1: Verhältnis der E/A Zeit an der Gesamtzeit bei serieller E/A

der E/A auf einem Rechenknoten deutlich langsamer steigen als auf 10 Rechenknoten. Der Anteil der E/A Zeit an der Gesamtzeit verdreifacht sich bei 160 Variablen und vervierfacht sich bei 320 Variablen. Die Ergebnisse der Skalierungsmessungen zeigen, dass die kürzeste Laufzeit bei drei oder vier Rechenknoten erreicht wird. Für alle Messungen wurden 10 Prozesse auf die Knoten verteilt. Die Gesamtlaufzeit sinkt ab dieser Anzahl von Rechenknoten nicht mehr ab. Es kann davon ausgegangen werden, dass die Berechnungszeit weiter sinkt, allerdings werden diese Gewinne von den steigenden E/A Zeiten wettgemacht.

6.2 Parallele E/A

Die Version von ECOHAM5 mit paralleler E/A ist die modifizierte Version. Die Messdaten sind in Abschnitt 5.3.3 und in Abschnitt 5.3.4 dargestellt. Die Werte zeigen, dass ECOHAM5 auch mit paralleler E/A von mehr Rechenknoten profitiert. Dabei schrumpft der Laufzeitvorteil von 50 Minuten bei 0 Variablen, auf 30 Minuten bei 160 Variablen und auf 16 Minuten bei 320 Variablen. Gleichzeitig zeigt sich, wie in Abbildung 6.2 dargestellt, dass es bei einem Rechenknoten eine lineare Steigerung im Verhältnis zwischen E/A Zeit und Gesamtzeit gibt. Je mehr Daten ausgegeben desto mehr Zeit wird auch

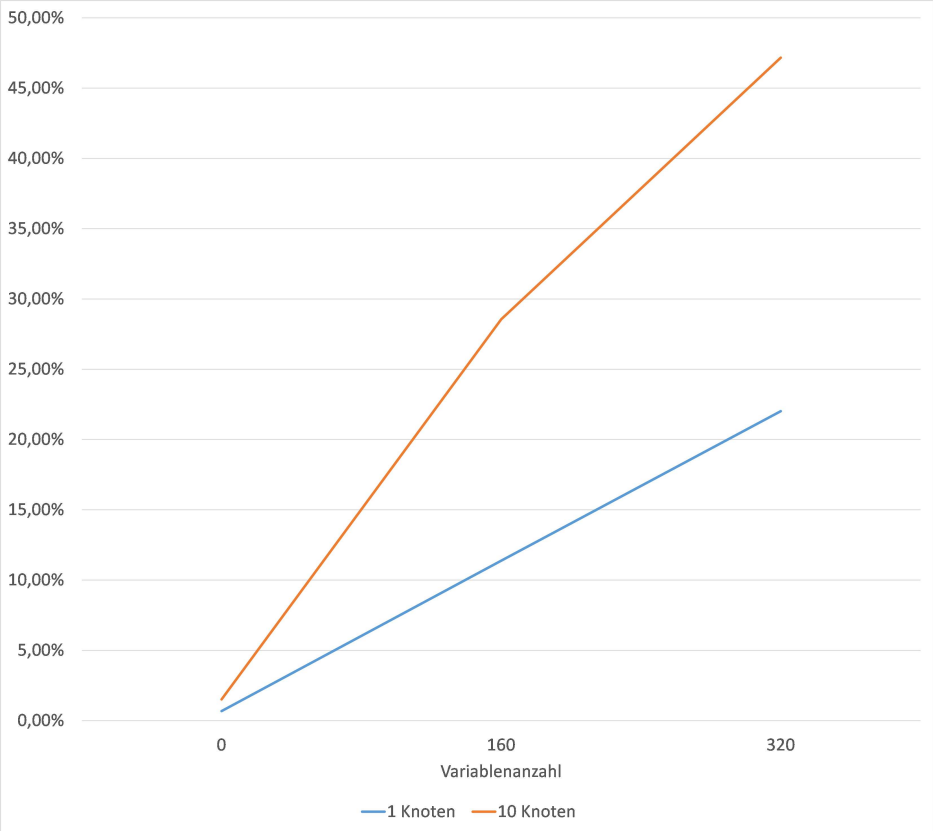

Abbildung 6.2: Verhältnis der E/A Zeit an der Gesamtzeit bei paralleler E/A

anteilig für die E/A aufgewendet. Bei doppelter E/A verdoppelt sich auch der Anteil an der Gesamtzeit. Bei 10 Rechenknoten ergibt sich ein ähnliches Bild, auch wenn dabei die Kurve bei 320 Variablen leicht abflacht. Die Steigung der Kurve ist bei 10 Rechenknoten höher als bei einem Rechenknoten. Wie in Abbildung 6.2 dargestellt ist, steigt der Anteil der E/A Zeit an der Gesamtzeit von einem auf 10 Rechenknoten an. Sowohl bei 160 Variablen als auch bei 320 Variablen ist der Anteil etwa das Zweifache oder das Zweieinhalbfache des Anteils auf einem Rechenknoten. Die Daten aus der Skalierungsmessung mit paralleler E/A (vergl. Abschnitt 5.3.4) zeigen, dass ECOHAM5 bei gleichbleibender Prozessanzahl auch mit paralleler E/A von einer steigenden Anzahl von Rechenknoten profitiert. Gleichzeitig steigt die Laufzeit der E/A bei mehr als einem Rechenknoten an. Die E/A Zeiten stabilisieren sich ab drei Rechenknoten, bei etwa 50 Minuten pro Prozess. Diese Stabilisierung zeigt, dass die parallele E/A einen hohen fixen Zeitanteil pro Prozess hat, dieser aber nicht von der Anzahl der Rechenknoten beeinflusst wird. Die Zeit, die in mit der Ausgabe in NetCDF verbracht wird, steigt von einem Rechenknoten auf zwei Rechenknoten stark an. Bei der weiteren Steigerung der Zahl der Rechenknoten stabilisiert sich auch diese Zeit um 3000 Sekunden.

6.3 Serielle und Parallele E/A

Im Vergleich der Gesamtzeit zwischen den zwei Versionen von ECOHAM5, lässt sich erkennen, dass die Version mit paralleler E/A sich nicht zu der Version mit serieller E/A verbessern kann. Nur auf einem Rechenknoten, ist, wie in Abbildung 6.3 zu sehen, die Laufzeit etwa gleich mit der Version mit paralleler E/A. Beim Vergleich der jeweiligen Konfigurationen mit 0% der Variablen lässt sich erkennen, dass es keine Verlangsamung des sonstigen Ablaufs durch das Einführen der parallelen E/A gab. Beide Versionen haben eine sehr ähnliche Gesamtlaufzeit. Bei der Betrachtung der Skalierungsmessungen zeigt sich, dass die Zeit für die parallelen E/A Operationen stark zunimmt. Insbesondere die Zeit, die konkret mit dem Schreiben der NetCDF Datei verbracht wird, steigt im Vergleich zwischen serieller und paralleler E/A stark an. Bei einem Knoten und 10 Prozessen steigt dieser Wert von 121 Sekunden auf 1268 Sekunden. Bei der parallelen E/A steigt dieser Wert auf über 3000 Sekunden an, was dem 30-Fachen der seriellen E/A entspricht. Gleichzeitig sinkt die Zeit, die nur in `eco_output_3D` verbracht wird. Bei der Version mit serieller E/A wurden auf 10 Rechenknoten 500 Sekunden in nur `eco_output_3D` verbracht, da hier die Übertragung auf den Masterknoten stattfindet. Bei der Version von ECOHAM5 mit paralleler E/A sinkt dieser Wert bei der gleichen Konfiguration auf 14 Sekunden ab, da in dieser Version keine Übertragung in `eco_output_3D` stattfindet.

Der Vergleich der Gesamtlaufzeiten, die in Abbildung 6.3 dargestellt sind, und der E/A Zeiten, welche in Abbildung 6.4 zu sehen sind, zeigt ein unterschiedliches Verhalten der zwei Versionen von ECOHAM5 auf. Wie erwähnt, ist nur die Gesamtlaufzeit auf einem Rechenknoten vergleichbar, bei zwei Knoten setzt sich die Version mit serieller E/A vor die Version mit paralleler E/A. Bei den E/A Zeiten wird deutlich, dass die parallele E/A deutlich mehr Zeit benötigt als die serielle E/A. Dabei liegt die Zeiten der beiden Versionen etwa um den Faktor 4 auseinander. Bei der seriellen E/A verdoppelt sich die Laufzeit der E/A von einem Knoten auf zwei Knoten und bleibt dann stabil bei diesem Faktor. Bei der parallelen E/A verdoppelt sich die Zeit von einem Knoten auf drei Knoten, danach bleibt sie nicht stabil und erreicht in der Spitze einen Faktor von 2,5 um dann wieder auf etwa Faktor 2 abzusinken. Insgesamt hat die parallele E/A eine hohe Ausführungszeit und dadurch einen deutlich höheren Anteil an der Gesamtlaufzeit.

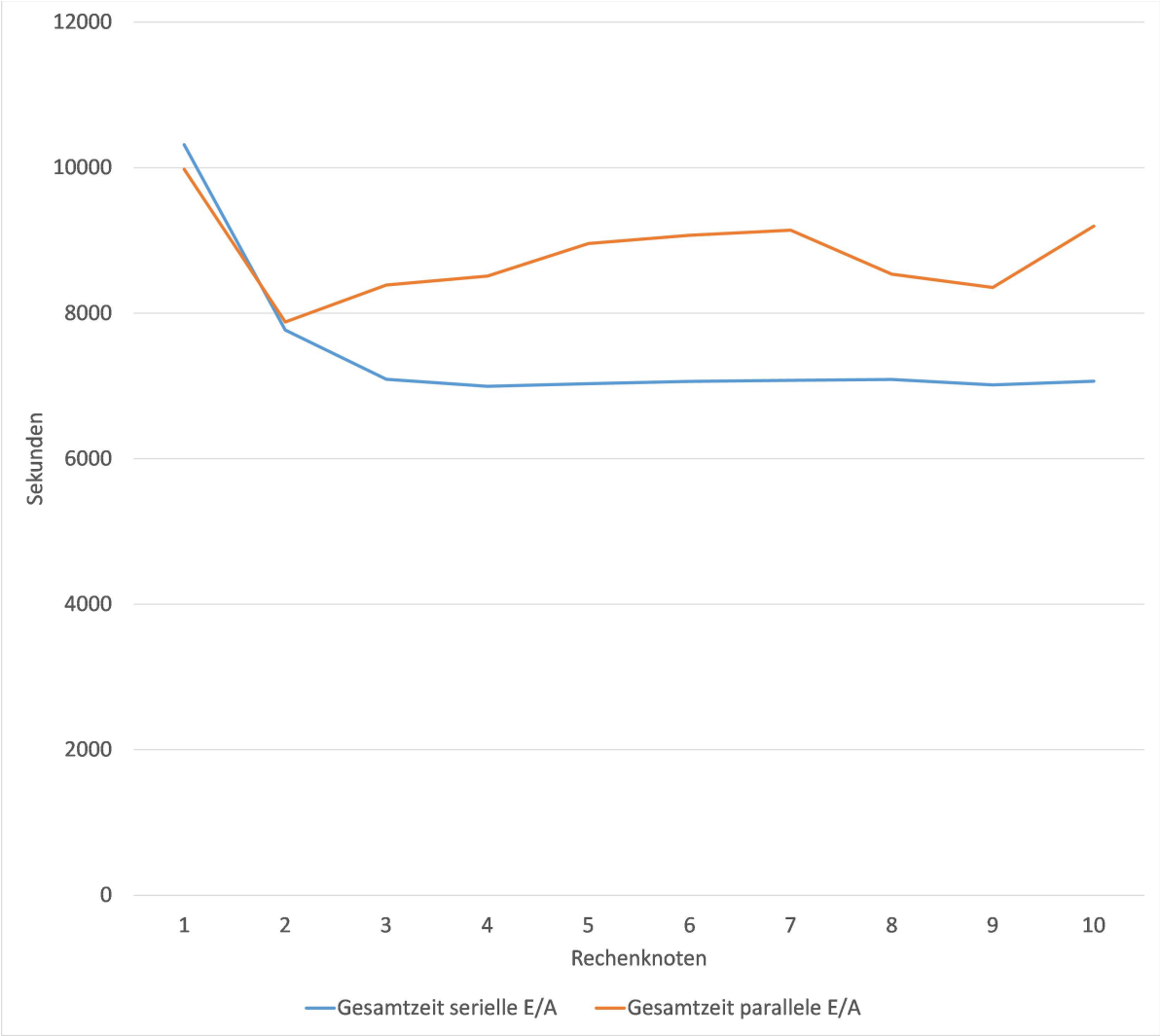

Abbildung 6.3: Gesamtlaufzeiten für serielle und parallele E/A

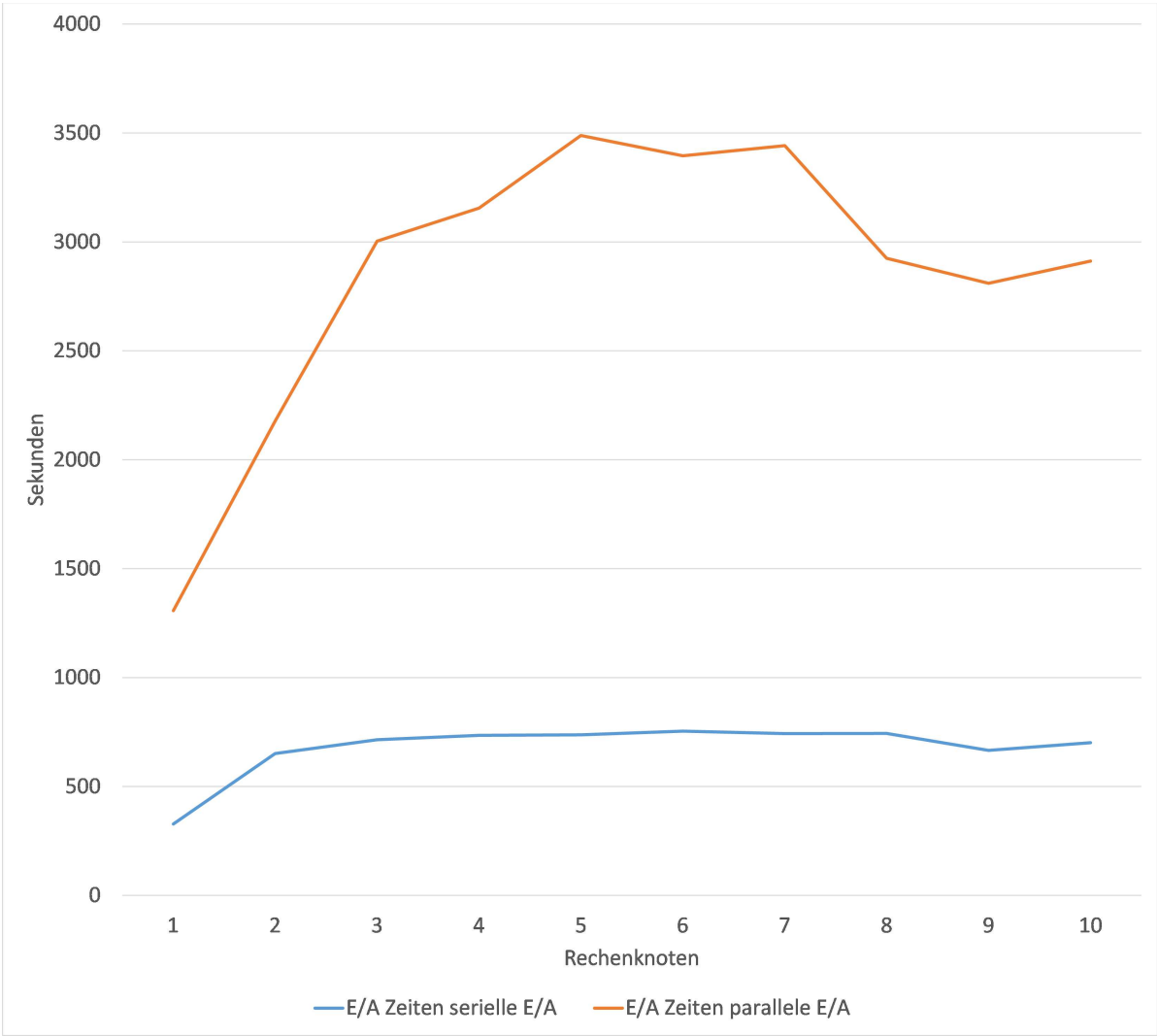

Abbildung 6.4: E/A-Zeiten für serielle und parallele E/A

7 Fazit

Das Ziel dieser Arbeit war die Analyse der Ein- und Ausgabe (E/A) des Ökosystemmodells ECOHAM5. ECOHAM5 ist ein paralleles HPC-Programm, das mit MPI parallelisiert ist. Es werden NetCDF Dateien als Ergebnis der Simulation ausgegeben. Wie bei vielen Erdsystem- und Klimamodelle wird bei ECOHAM5 nur serielle E/A durchgeführt, was die Skalierung stark einschränkt. Diese Einschränkung wird auf zukünftigen HPC-Systemen zunehmen. Für die Analyse wurde ECOHAM5 für parallele E/A erweitert und es wurde die Performance gemessen und analysiert. Zudem wurde parallele E/A in ECOHAM5 implementiert.

Grundlagen ECOHAM5 ist ein Erdsystemmodell, das die Ökologie der Nordsee simuliert. Das Modell wird genutzt um Fragen des Kohlenstoffflusses in der Nordsee im Rahmen des Klimawandels (Lorkowski et al., [LPMK12]; Pätsch und Kühn, [PK08]) sowie Fragen zur Auswirkung unterschiedlicher Belastung des Ökosystems Nordsee durch Nährstoffeinträge von Stickstoff und Phosphor (Lenhart et al. [LMBB$^+$10, LDG$^+$13]) zu untersuchen. Dazu wird die Nordsee in ein dreidimensionales Gitter unterteilt und für jede Gitterzelle werden für eine Reihe von Zustandsvariable numerische Differenzialgleichungen gelöst. Das Modellgebiet des ECOHAM-Gitters umfasst den Nordwesteuropäischen Kontinentalschelf (NECS) und Teile des angrenzenden Nordostatlantiks (Abbildung 4.1). ECOHAM5 ist in Fortran implementiert und nutzt MPI für die parallele Ausführung mit mehreren Prozesseen. Jeder dieser Prozesse ist an der Berechnung der Simulation beteiligt. Die Simulationsergebnisse werden in der ursprünglichen Version von ECOHAM5 von einem Prozess/Rechenknoten dem Masterknoten mit NetCDF gespeichert. Diese serielle E/A wurde in dieser Arbeit verschiedentlich untersucht. Die Implementierung wurde statisch anhand des Quellcodes analysiert. Die Ausführung wurde gemessen und mithilfe des Tracingprogramms Vampir/Score-P ausgewertet. Für die E/A nutzt ECOHAM5 die Bibliotheken MPI, MPI-IO, HDF5 und NetCDF.

Evaluation Für die Evaluation wurde ein Referenzlauf definiert. Dieser definiert, dass nur 3D-Variablen ausgegeben werden und dass ein Programmablauf 6 Monate simuliert. Bei der Analyse der Messdaten zur seriellen E/A wurde ein möglicher Engpass in der Ausgabe festgestellt. Alle Daten werden doppelt übertragen, einmal auf den Masterknoten und dann in das verteilte Dateisystem Lustre. Als mögliche Optimierung wurde parallele E/A in ECOHAM5 für die Ausgabe der Simulationsergebnisse eingeführt. Die Implementierung der parallelen E/A nutzt NetCDF-4 Funktionalität um die parallele E/A zu ermöglichen. Parallele E/A in NetCDF-4 ist mit HDF5 umgesetzt, welches wiederum MPI-IO nutzt. Die Gesamtlaufzeit von ECOHAM5 mit paralleler E/A ist nicht

schneller als die ursprüngliche Version. Die Performance der parallelen E/A sind absolut bis zu Faktor 3 langsamer als die serielle E/A. In der Konfiguration mit zehn Rechenknoten werden etwa 45% der Zeit mit paralleler E/A verbracht, bei der seriellen E/A werden im vergleichbaren Programmablauf nur etwa 16% der Zeit für E/A aufgewendet. Gleichzeitig zeigt sich bei der doppelt so großen Ausgabedateien, dass sich die Laufzeit der parallelen E/A dabei weniger als verdoppelt. Im Vergleichsszenario mit serieller E/A ist die Steigerung der Laufzeit mehr als Faktor 2. Insgesamt kann aber kein Vorteil für die parallele E/A in ECOHAM5 festgestellt werden.

Ausblick Die in dieser Arbeit untersuchte Implementierung von ECOHAM5 nutzt Collective-I/O für die parallele E/A. Eine Untersuchung einer Implementierung mit Independet-I/O könnte interessant sein, da hierbei möglicherweise parallele E/A mit weniger Overhead durchgeführt werden kann. Eine weitere Möglichkeit für Folgearbeiten könnte eine Betrachtung von ECOHAM5 mit mehr Prozessen und mehr Rechenknoten sein. Um dies zu unterstützen müsste ECOHAM5 für mehr Prozesse erweitert werden. Auch könnte es sinnvoll sein, das E/A Muster von ECOHAM5 in einem Bechmarktool wie IOR nachzustellen um damit einfacher Tests durchführen zu können.

Literaturverzeichnis

[Amd67] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967. 1.1

[Bar14] Christopher Bartz. An in-depth analysis of parallel high level i/o interfaces using hdf5 and netcdf-4. mathesis, University of Hamburg, April 2014. 3, 5.3.5

[BCK+15] Christopher Bartz, Konstantinos Chasapis, Michael Kuhn, Petra Nerge, and Thomas Ludwig. A best practice analysis of hdf5 and netcdf-4 using lustre. In *High Performance Computing*, pages 274–281. Springer, 2015. 3, 3

[DKR16] DKRZ. Klimamodellierung. Website, 2016. 1.2

[DLP03] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003. 1.1

[fM16] Max-Planck-Institut für Meteorologie. Icon (icosahedral non-hydrostatic) general circulation model. Website, 2016. 1.2

[GGK+16] Fabian Große, Naomi Greenwood, Markus Kreus, Hermann-Josef Lenhart, Detlev Machoczek, Johannes Pätsch, Lesley Salt, and Helmuth Thomas. Looking beyond stratification: a model-based analysis of the biological drivers of oxygen deficiency in the north sea. *Biogeosciences*, 13(8):2511–2535, 2016. 4.1.1

[Gro16] The HDF Group. Hdf5 users guide. Website, 2016. 2.2, 2.2, 1, 2.3, 2.2, 7

[How12] Mark Howison. Tuning hdf5 for lustre file systems. In *Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10)„ Heraklion, Crete, Greece, September 24, 2010*, 2012. 3

[IG13] The IEEE and The Open Group. The open group base specifications issue 7 - pread, read - read from a file. Website, 2013. 2.2

[IGC+14] Florin Isaila, Javier Garcia, Jesus Carretero, RB Ross, and Dries Kimpe. Making the case for reforming the i/o software stack of extreme-scale systems. *Preprint ANL/MCS-P5103-0314, Argonne National Laboratory*, 2014. 3

[Int16a] Intel. Intel® xeon® processor x5650. Website, 2016. 5.2.2

[Int16b] Intel and Oracle. *Lustre * Software Release 2.x Operations Manual*, 2016. 2.4, 2.5, 7

[KHKS08] Rainer Keller, Valentin Himmler, Bettina Krammer, and Alexander Schulz. *Tools for High Performance Computing: Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart*. Springer Science & Business Media, 2008. 2.5

[Lab16] Argonne National Laboratory. Romio. Website, 2016. 2.1.1

[LDG+13] H Lenhart, X Desmit, F Große, D Mills, G Lacroix, H Los, A Ménesguen, J Pätsch, T Troost, J van der Molen, et al. Report on "distance to target" modelling assessment by icg-emo. *OSPAR Eutrophication series*, (599), 2013. (document), 4.1.1, 7

[LMBB+10] Hermann-J Lenhart, David K Mills, Hanneke Baretta-Bekker, Sonja M Van Leeuwen, Johan Van Der Molen, Job W Baretta, Meinte Blaas, Xavier Desmit, Wilfried Kühn, Geneviève Lacroix, et al. Predicting the consequences of nutrient reduction on the eutrophication status of the north sea. *Journal of Marine Systems*, 81(1):148–170, 2010. (document), 4.1.1, 7

[LPMK12] Ina Lorkowski, Johannes Pätsch, Andreas Moll, and Wilfried Kühn. 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*, 100:38–57, mar 2012. (document), 4.1.1, 7

[MPI16] MPICH. Mpich. Website, 2016. 2.1

[MRP+15] Cyriel Minkenberg, German Rodriguez, Bogdan Prisacari, Laurent Schares, Philip Heidelberger, Dong Chen, and Craig Stunkel. Large-scale system partitioning using ocs. In *Photonics in Switching (PS), 2015 International Conference on*, pages 235–237. IEEE, 2015. 1.1

[NCO16] NCO. Nco homepage. Website, May 2016. 2.3

[Ope] OpenMPI. Open mpi. Website, 2016. 2.1

[Ope16] OpenSFS. Lustre 2.8.0 released at lug 2016 and declared general availability. Website, April 2016. 1.3

[Pie16] David W. Pierce. Ncview: a netcdf visual browser. Software, 1995-2016. 2.6.1, 5.3.1

[PK08] Johannes Pätsch and Wilfried Kühn. Nitrogen and carbon cycling in the north sea and exchange with the north atlantic—a model study. part i. nitrogen budget and fluxes. *Continental Shelf Research*, 28(6):767–787, apr 2008. (document), 4.1.1, 7

[RDE+16] R Rew, G Davis, H Emmerson, E Hartnett, and D Haimbigner. The netcdf users guide. university corporation for atmospheric research, 2016. 2.3, 2.4, 7

[SSL+13] Seung Woo Son, Saba Sehrish, Wei-keng Liao, Ron Oldfield, and Alok Choudhary. Dynamic file striping and data layout transformation on parallel system with fluctuating i/o workload. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013. 3

[TBC09] Dilek Topcu, Uwe Brockmann, and Ulrich Claussen. Relationship between eutrophication reference conditions and boundary settings considering OSPAR recommendations and the water framework directive—examples from the german bight. *Hydrobiologia*, 629(1):91–106, may 2009. 4.1.1

[TGL99] Rajeev Thakur, William Gropp, and Ewing Lusk. Data sieving and collective i/o in romio. In *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, pages 182–189. IEEE, 1999. 2.1.1

[Top16] Top500. The linpack benchmark. Website, 2016. 1.1

[Tro15] Stephen J Trowbridge. Ethernet and otn-400g and beyond. In *Optical Fiber Communications Conference and Exhibition (OFC), 2015*, pages 1–18. IEEE, 2015. 1.1

[Uni11] Unidata. ncdump. Website, 2011. 2.6.3

[WG99] Ewing Lusk William Gropp, Rajeev Thakur. *Using Mpi-2: Advanced Features of the Message Passing Interface*. MIT PR, 1999. 2.1

[Wik16a] Wikipedia. Thread (informatik) — wikipedia, die freie enzyklopädie, 2016. [Online; Stand 6. August 2016]. 1.1

[Wik16b] Wikipedia. Vorsätze für maßeinheiten — wikipedia, die freie enzyklopädie, 2016. [Online; Stand 15. Juni 2016]. 1

[Zie15] Remik Ziemlinski. nccmp - compare netcdf files. Software, 2004-2015. 2.6.2, 5.3.1

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Programm-Listings

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Master of Science Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin damit einverstanden, dass meine Abschlussarbeit in den Bestand der Fachbereichsbibliothek eingestellt wird.

Ort, Datum	Unterschrift