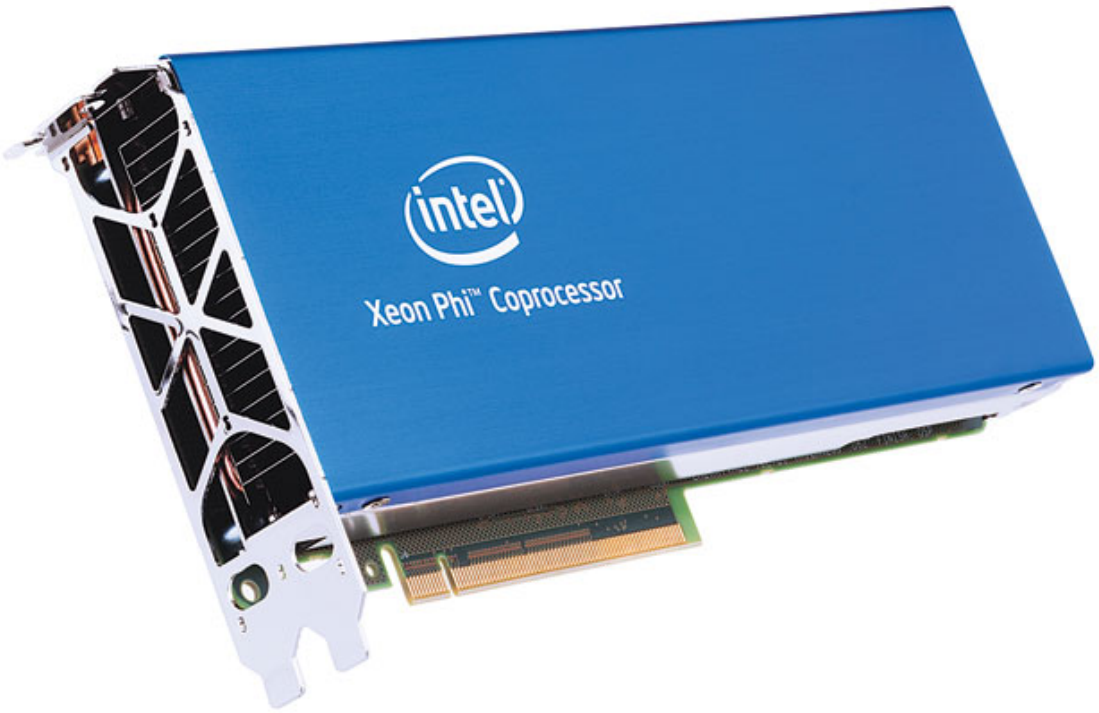
Einsatz von Beschleunigerkarten für das Postprocessing großer Datensätze

— Bachelorarbeit —

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

Vorgelegt von:	Jannek Squar
E-Mail-Adresse:	1squar@informatik.uni-hamburg.de
Matrikelnummer:	6357990
Studiengang:	Computing in Science
Erstgutachter:	Prof. Dr. Thomas Ludwig
Zweitgutachter:	Dr. Frank Heitmann
Betreuer:	Petra Nerge, Michael Kuhn

Hamburg, den 03.12.2014



Intel Xeon Phi Coprocessor 5110P[Cou]

Abstract

Diese Bachelorarbeit beschäftigt sich mit der Frage, ob der Einsatz von Beschleunigerkarten einen Vorteil beim Postprocessing großer Datensätze mit sich bringt. Diese Fragestellung wird anhand einer Xeon-Phi-Karte und dem Programm *Harmonic Analysis* untersucht, welches auf den Ausgabedaten einer Ozeansimulation eine harmonische Analyse durchführt.

Zunächst werden die herausragenden Merkmale und unterschiedlichen Betriebsmodi - der native Modus, der Offload-Modus und der symmetrische Modus - der Xeon-Phi-Karte vorgestellt; auch der Aufbau von *Harmonic Analysis* wird näher beschrieben, um Einsatzmöglichkeiten der Xeon-Phi-Karte zu klären. Dabei zeichnen sich erste Probleme ab, da die verschiedenen Betriebsmodi Anpassungen an verwendeten Bibliotheken erforderlich machen. *Harmonic Analysis* wird dann zunächst so überarbeitet, dass Teile des Programms im Offload-Modus auf die Karte geladen und dort ausgeführt werden, außerdem wird die Möglichkeit der Vektorisierung geprüft, da die Kerne der Xeon-Phi-Karte jeweils über eine große Vektoreinheit verfügen.

Bei der Leistungsanalyse wird die Programmlaufzeit für unterschiedliche Startparameter verglichen, im Endeffekt muss aber festgestellt werden, dass sich die Verwendung der Xeon-Phi-Karte für *Harmonic Analysis* nicht rentiert hat, da die erzielte Leistung hinsichtlich Effizienz, Kosten und absoluter Verbesserung der Laufzeit bei Einsatz der Xeon-Phi-Karte schlechter ist, als wenn sie nicht verwendet wird. Da aber im Rahmen dieser Bachelorarbeit noch nicht alle Möglichkeiten ausgereizt worden sind, werden noch mögliche Ansatzpunkte zur Weiterarbeit aufgeführt.

Inhaltsverzeichnis

1. Einleitung

Seit einigen Jahren bietet Intel eine neue Beschleunigerarchitektur an - Larrabee. Darauf basierend verkauft Intel seit 2012 Xeon-Phi-Coprozessoren. Bei diesen Xeon-Phi-Karten handelt es sich streng genommen um Erweiterungskarten, im Allgemeinen bezeichnet man die Xeon-Phi-Karte aber dennoch als Beschleunigerkarte. Das Ziel dieser Bachelorarbeit ist es, das Potential dieser Beschleunigerkarte im Bereich des Hochleistungsrechnen zu untersuchen. Ein Anwendungsfall des Hochleistungsrechnens ist das Durchführen von Simulationen der Klimaforschung.

In der Vergangenheit wurden Erkenntnisse durch häufige Wiederholung von Experimenten oder gar durch schieren Zufall gewonnen. Heutzutage ist dieser Ansatz in einigen Bereichen nicht mehr anwendbar, da dieses Vorgehen zu viele Ressourcen (Mensch oder Material) beansprucht oder sogar nicht anwendbar ist, wenn der Maßstab des Fachgebietes zu extrem ist - wie zum Beispiel in der Quantenmechanik, Astronomie oder Klimaforschung. Dann können zwar noch immer Erkenntnisse auf Grundlage theoretischer Überlegungen gewonnen werden, oft verhindert aber der Umfang der Theorie oder sogar Lücken darin eine analytisch korrekte Auswertung. In diesem Fall kann man sich aber noch immer mit Simulationen behelfen. Um eine hohe Präzision dieser Wirklichkeitsabbildung zu erreichen, ist es notwendig die Simulation länger und eventuell auch auf größeren Datensätzen laufen zu lassen, deren Verarbeitung aber auch zusätzliche Arbeitsschritte erfordern. Zum einen müssen die Daten an das Eingabeformat der Simulation angepasst werden, bevor auf ihnen gerechnet werden kann. Zum anderen ist es notwendig, die Ergebnisse mittels Postprocessing anzupassen, damit wir Menschen die Ergebnisse auch vollständig aufnehmen und interpretieren können.

Eine Verbesserung der Laufzeit solcher Simulationen kann erreicht werden, indem die Rechenleistung verbessert wird. Da die Superrechner der aktuellen Generation ihre Leistung hauptsächlich durch parallel arbeitende CPUs beziehen, kann dies zum Beispiel dadurch erreicht werden, dass die CPU-Taktraten verbessert oder weitere CPUs hinzugeschaltet werden. Wie in [Man11] dargestellt wird, ist eine Steigerung der CPU-Taktrate allerdings problembehaftet, unter anderem ergeben sich folgende Probleme:

1. Quanteneffekte: Um die Taktfrequenz der CPU bei gleichbleibender Fläche zu vergrößern, müssen mehr Bauelemente (wie z.B. Transistoren) auf konstanter Fläche untergebracht werden. Neben aufwendigeren Konstuktionsverfahren müssen unter einer Strukturgröße von 11nm allerdings auch Quanteneffekte berücksichtigt werden, die bei einem solch kleinen Maßstab an Bedeutung gewinnen. Die Reduktion der Bauteilgröße führt dazu, dass die Isolationsschichten der Transistoren nur noch

wenige Atomschichten dick ist, wodurch die Wahrscheinlichkeit von Tunneleffekten ansteigt. Dabei tunneln Elektronen dann durch die Barriere eines gesperrten Transistors und bewirkt so falsche Ergebnisse. Eine mögliche Lösung sind Verfahren zur Fehlererkennung und Neuberechnung oder der Einsatz alternativer Materialien für Transistoren wie z.B. Molybdänit oder Graphen, die die Tunnelwahrscheinlichkeit reduzieren, von denen aber erst Prototypen existieren.

2. Wärmeentwicklung: Eine höhere CPU-Taktung bedeutet mehr Schaltvorgänge, wodurch mehr Wärme umgesetzt wird [EK]. Um weiterhin die richtige Arbeitstemperatur aufrechtzuerhalten, sind zusätzliche Kühlmaßnahmen erforderlich. Da praktisch nicht beliebig viel Wärme abgeführt werden kann, liegt die derzeit maximal erreichbare Taktung bei 8GHz [Chi11].

Diese Probleme vermeidet man, wenn die zusätzliche Rechenleistung durch Parallelschaltung der Recheneinheiten erzielt wird. Das Grundgerüst von Beschleunigerkarten (wie zum Beispiel Grafikkarten) basiert auf genau diesem Prinzip, je nach Bauart können sich hunderte/tausende von parallel arbeitenden Recheneinheiten auf einer einzigen Beschleunigerkarte befinden. Daher sind Beschleunigerkarten üblicherweise der aktuellen Prozessorgeneration überlegen, wenn es darum geht, auf einen großen Datenvektor eine Operation anzuwenden (SIMD[1]). Da die meisten Beschleunigerkarten allerdings eine eigene Programmierumgebung (z.B. Cuda bei NVIDIA-Karten) benötigen, können Programme, die vielleicht bereits seit Jahrzehnten aufgebaut und benutzt werden, nicht ohne Änderungen auf ihnen laufen[BR]. Änderungen an diesen Programmen können allerdings weitere Anpassungen notwendig machen, wenn sie Grundlage für weitere Programme sind oder selbst anderer Programme bedürfen. Schnell kommt man an den Punkt, dass solche Änderungen nicht durchführbar sind, wenn sie manuell von einem Menschen gemacht werden müssen.

An dieser Stelle verspricht Intel eine mögliche Lösung: Ihre Intel Many-Integrated-Core-Architektur (Intel MIC, verbesserte Larrabee-Architektur), auf der die Xeon-Phi-Karte basiert, wird mit einem Compiler ausgeliefert, der es ermöglichen soll, bereits bestehende Software, die in C/C++/Fortran geschrieben wurde, auf der Xeon Phi auszuführen, ohne dass manuell umfassende Änderungen am Programm vorgenommen werden müssen.

1.1. Motivation

Unter Postprocessing im Bereich des Hochleistungsrechnen versteht man die Aufbereitung von Daten, die man aus durchgeführten Simulationen erhält, und wird dabei zumeist mit einem separaten Programm ausgeführt. Der Zweck dieser Aufbereitung besteht darin, die Ausgabe der Simulation an die Bedürfnisse von Menschen oder Programmen anzupassen, die mit den Simulationsergebnissen weiterarbeiten oder diese interpretieren wollen. Die Anpassung könnte beispielsweise darin bestehen, den großen Ausgabe-Datensatz der

[1] Single Instruction Multiple Data

Simulation in eine grafische Darstellung umzuwandeln, damit Menschen das berechnete Ergebnis auch ohne großen Aufwand wahrnehmen können [Wikc]. Oder wenn nicht alle Ergebnisse der Simulation für den eigenen Zweck relevant sind, könnten die benötigten Daten aus dem Output rausgefiltert werden - besonders die Verwendung von Dritt-Programmen macht Postprocessing notwendig, da die Simulation dann nicht an die eigenen Bedüfnisse angepasst ist und auch nicht-benötigte Daten liefert. Auf den Simulationsergebnissen können so auch weitere Rechnungen angestellt werden, um aus der Vorlage weitere Erkenntnisse abzuleiten, für deren Findung die Simulation selbst nicht zuständig war. Ein Beispiel für diese Form des Postprocessings wird in dieser Bachelorarbeit im Kapitel *Programmbeschreibung* (Seite 28) beschrieben.

Besonders in der Klimaforschung steht Postprocessing an der Tagesordnung: Die Größenordnung des Klimas und damit auch die Größe der benutzten Datensätze erfordert für eine möglichst genaue Beschreibung das Zusammenspiel verschiedener Programme, die Daten in unterschiedlichen Formaten verarbeiten, und dass am Ende die relevanten Daten herausgefiltert werden. Die voranschreitende Entwicklung von Speichermedien und Geräten zur Langzeit-Messung, wie z.B. mit Satelliten, Flugzeugen, oder Netzwerken von Sensoren, ermöglicht es Klimaforschern, umfangreiche Datensätze zu sammeln [ET01]. Um die Ergebnisse der Rechnungen auf diesen Daten hierbei auch in vertretbarer Zeit zu erhalten, benötigt man eine den Anforderungen entsprechende Rechenleistung. Das DKRZ[2] in Hamburg betreibt hierzu neben kleineren Rechenclustern auch den Superrechner „blizzard“, der mit einer Spitzenleistung von 158 TFLOPS eine solide Arbeitsplattform für Klimawissenschaftler bietet. Insgesamt arbeiten in diesem Superrechner u.a. 8448 Kerne [DKRb]. Anstelle von parallel geschalteten CPUs könnte man diese hohe Zahl aber auch relativ schnell mit parallel arbeitenden Xeon-Phi-Karten erreichen, da eine einzelne Karte rein rechnerisch mit ihren 60 Kernen eine Leistung von bis zu 1008 GFLOPS erreichen kann (siehe auch *Technische Daten*, Seite 10). Zumindest könnte man die Xeon Phi unterstützend einsetzen. Das Ziel dieser Bachelorarbeit ist es zu prüfen, inwiefern die hochparallelisierte Xeon Phi einen möglichen Ersatz für CPUs darstellt und welche Zielgruppe tatsächlich von der Xeon Phi profitieren kann.

1.2. Übersicht

Als Grundlage für die Untersuchung der Xeon Phi verwendet der Autor das Programm *Harmonic Analysis* welches von Petra Nerge (Arbeitsbereich *Wissenschaftliches Rechnen*, Universität Hamburg) geschrieben worden ist. Dieses Programm führt Postprocessing auf dem Output-Datensatz von *MPIOM*[3] durch. *MPIOM* wurde am Max Planck Institut für Meteorologie in Hamburg entwickelt und ist aktuell Bestandteil des Erdsystemmodells *MPIESM* [4], welches das Zusammenspiel verschiedener Klimafaktoren (wie z.B. die Vegetationsdynamik oder den Energieaustausch zwischen Atmosphäre, Land und Ozean) be-

[2]Deutsches Klimarechenzentrum
[3]Max-Planck-Institute Ocean Model
[4]Max Planck Institute Earth System Model

schreibt, um so Rückschlüsse auf das Klima und dessen Entwicklung ziehen zu können (siehe auch [Max]). *MPIOM* ist dabei u.a. für die Simulation der Ozeanbewegungen zuständig.

Harmonic Analysis erhält als Eingabe den Output-Datensatz von *MPIOM* und führt auf der zeitlichen und räumlichen Entwicklung der Meeresstände eine harmonische Analyse durch, um aus diesen Daten die einzelnen Perioden der Gezeiten herauszurechnen. Dabei wird der Einfluss der einzelnen Perioden auf die resultierende Gezeit sichtbar und lässt Rückschlüsse darauf zu, welche natürlichen Faktoren welchen Einfluss auf die Entstehung der Gezeiten haben. Eine detailliertere Beschreibung von In- und Output von *Harmonic Analysis* erfolgt im Abschnitt *Verwendungszweck*, Seite 28.

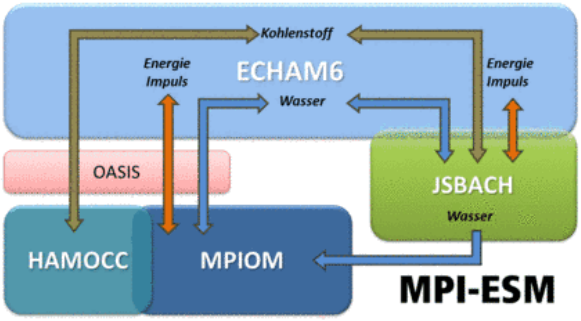


Abbildung 1.1.: Aufbau von MPIESM [DKRa]

1.3. Geplantes Vorgehen

Um einen Eindruck von der Xeon Phi zu erhalten, werden im nächsten Kapitel *Beschreibung der Intel Xeon Phi* zunächst einmal die technischen Eckdaten der Karte vorgestellt, um im darauffolgenden Abschnitt *Benutzung der Intel Xeon Phi* auf die Benutzung der Xeon-Phi-Karte einzugehen. Im Zuge dieser Beschreibung wird der Autor auch auf Anpassungen der Software auf dem Host eingehen, die für eine korrekte Programmausführung auf der Xeon-Phi-Karte notwendig sind. Nach dieser Vorstellung wird in *Programmbeschreibung* das Programm *Harmonic Analysis* näher beschrieben: Dabei wird auf den mathematischen Hintergrund und den Programmaufbau und -ablauf eingegangen, um dann in Kapitel *Programmanpassungen* das Programm an die Xeon Phi anzupassen und mithilfe der Karte auszuführen. Abschließend wird der Effekt der unternommenen Programmanpassungen mittles einer Leistungsanalyse dargestellt und bewertet.

2. Beschreibung der Intel Xeon Phi

Der Zweck dieses Kapitels ist es, dem Leser einen Überblick über die Xeon Phi zu geben, indem zunächst die Hardware der Xeon-Phi-Karte vorgestellt wird. Der Übersicht halber, wird die Beschreibung aber nicht auf alle Details dieser Hardware eingehen - für eine vollständige Beschreibung der technischen Funktionsweise der Xeon-Phi-Karte sei auf die nachfolgenden Quellen verwiesen. Die genaue Bezeichnung der verwendete Xeon-Phi-Karte lautet Intel© Xeon Phi™ Coprocessor 5110P, der Einfachheit halber wird sie aber in dieser Bachelorarbeit mit Xeon-Phi-Karte referenziert.

Als Hersteller bietet Intel im Internet unterschiedliche Arten der Dokumentationen für die Xeon-Phi-Karte an. Auf [Inti] bietet Intel Material an, damit man sich einen schnellen Überblick verschaffen und zu einer ersten Einschätzung der Eignung der Karte für die eigenen Zwecke gelangen kann. Außerdem gibt es dort eine Sammlung von Bibliotheken und Tools, die beim Aufsetzen eines frisch aufgesetzten Systems hilfreich sein könnten - u.a. werden Compiler, Debugger sowie Verwaltungs- und Analyseprogramme angeboten. Wird die Xeon-Phi-Karte auf einem bereits eingerichteten System eingesetzt, sind die meisten Tools oder entprechende Alternativen aber vermutlich bereits vorhanden. Für eine tiefergehende Beschäftigung mit der Xeon-Phi-Karte bietet Intel zudem Anleitungen und auch ganze Video-Serien an. Interessenten an weiteren ausführlichen Beschreibung der Xeon-Phi-Karte seien auch auf zwei weitere Quellen verwiesen: zum einen das Buch [WZS+14], welches Mitarbeiter des *Inspur-Intel China Parallel Computing Joint Lab* geschrieben haben, sowie eine Online-Anleitung [BBI+] der europäischen Initiative *PRACE*[1]

2.1. Technische Daten

Dies ist eine Übersicht über die wesentlichen technischen Daten der Xeon-Phi-Karte, eine erweiterte Übersicht ist auf Intels Website einsehbar: [Intg]. Im nächsten Abschnitt *Zusammenspiel der Bauelemente* (Seite 11) wird die Xeon-Phi-Karte noch näher beschrieben.

[1] Partnership for advanced computing in Europe, `http://www.prace-ri.eu/` (Zuletzt aufgerufen: 28.11.2014)

Typbezeichnung	5110P
Anzahl Kerne	60
Taktfrequenz	1,053 GHz
Befehlssatz	64bit
L2-Cache	30 MB
RAM	8 GB
Max. Speicherbandbreite	320 GB/s
PCIe-Version	2.0

Tabelle 2.1.: Technische Eckdaten der Xeon-Phi-Karte

2.1.1. Zusammenspiel der Bauelemente

Der Intel Xeon Phi Coprocessor ist kein CPU-Ersatz sondern ist über PCI Express mit dem Motherboard des Hosts verbunden, der somit über eine eigene CPU verfügt. Die Kommunikation zwischen Xeon-Phi-Karte und Host erfolgt über eine Peer-to-Peer-Verbindung über PCI Express. Es können in ein Hostsystem auch mehrere Xeon-Phi-Karten gleichzeitig eingebaut werden, die untereinander ebenfalls über Peer-to-Peer miteinander kommunizieren können, ohne einen Umweg über den Host gehen zu müssen. So können auch mehrere Xeon-Phi-Karten, räumlich getrennt auf verschiedenen Hostsystemen, über die jeweilige Netzwerkkarte Verbindung zueinander aufnehmen. Außerdem läuft auf der Xeon-Phi-Karte ein eingebettetes Linux-Betriebssystem, Benutzer können sich über die PCIe-Verbindung also mittels `ssh` auf der Xeon-Phi-Karte einloggen und wie einen Netzwerkknoten verwenden. Das Betriebssystem hat aber nicht den vollen Umfang einer vollwertigen Linux-Distribution, da sich die gebotene Funktionalität auf den Umfang der zugrundeliegenden Tool-Sammlung Busybox beschränkt (mehr zu Busybox unter [Bus]), und wird daher auch als Mikro-Linux bezeichnet. Zumindest kann die Xeon-Phi-Karte dadurch aber pThreads und andere Parallelisierungskonzepte unterstützen.

Die Hauptbestandteile einer Xeon-Phi-Karte sind die die Kerne, Caches, GDDR5-Module sowie ein bidirektionales Ring-Bussystem.

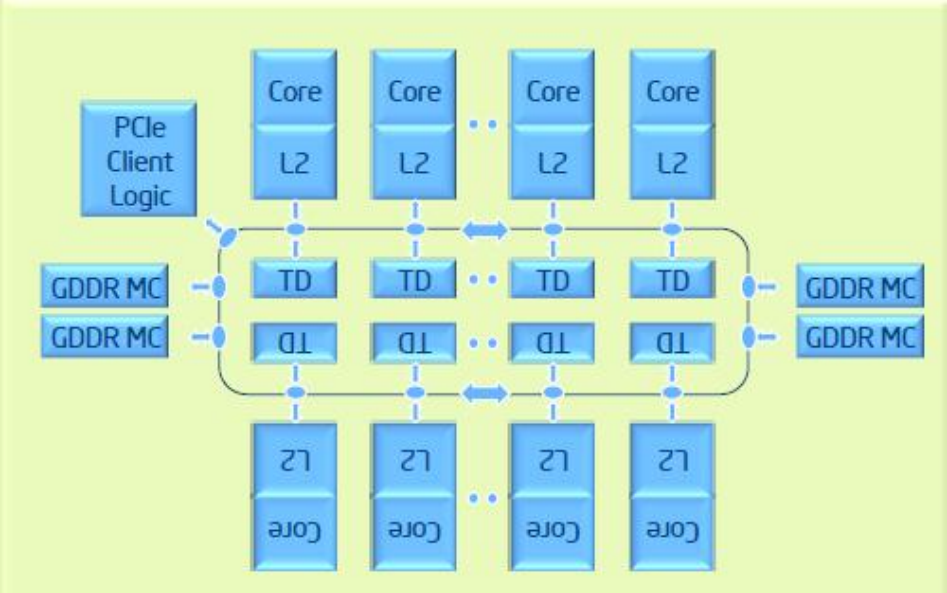


Abbildung 2.1.: Aufbau der Xeon-Phi-Karte [Intf]

Auf der Xeon-Phi-Karte befinden sich 60 Kerne à 1,053GHz, die eine verbesserte Version eines alten Pentium-Modells (P54C) sind. Die Leistungsaufnahme der Kerne ist dabei gering genug, dass eine Passivkühlung ausreicht. Jeder Kern ist mit einem L2-Cache (512kB) an das Ring-Bussystem angeschlossen und unterstützt bist zu vier Hardwarethreads. Besonders erwähnenswert ist dabei die eingebaute 512-Bit-Vektoreinheit, die pro Zyklus 16 Operationen mit einfacher Genauigkeit oder 8 Operationen mit doppelter Genauigkeit ausführen kann. Mittels Multiply-Accumulate-Befehlen (MAC) kann die Recheneinheit in einer Instruktion sowohl eine Addition als auch eine Multiplikation auf einmal ausführen. Werden alle 60 Kerne voll ausgelastet, indem alle Vektoreinheiten jeweils 8 MAC-Operationen mit doppelter Rechengenauigkeit ausführen, ergibt dies im normalen Betrieb die maximale Leistung von ca. 1 TFLOPS, die man auf der Xeon-Phi maximal erreichen kann [FVS+][BR]:

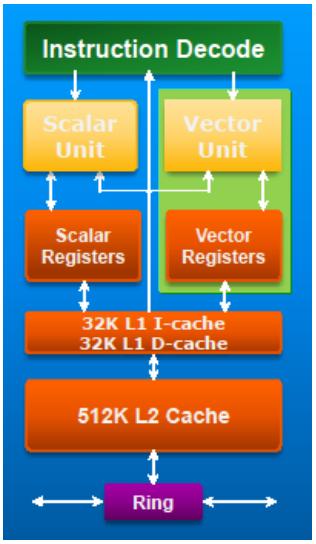


Abbildung 2.2.: Aufbau eines Kerns [Intc, S. 11]

$$60\ \texttt{Kerne} \cdot 1,05\ \texttt{GHz} \cdot 4\ \texttt{Threads} \cdot 8\ \texttt{Operationen} = 1008\ \texttt{GFLOPS}$$

Auf der Xeon-Phi-Karte stehen insgesamt 8GB GDDR5 an Arbeitsspeicher zur Verfügung, dabei handelt es sich um die neuste Generation von DDR-Speicher, der für

Grafikanwendungen ausgelegt ist (zum Vergleich: in der aktuellen Next-Gen-Konsole *Playstation 4* sind ebenfalls 8GB GDDR5 verbaut [PS4])[GDD].

Für eine noch detaillierte Einsicht in die Hardware der Xeon-Phi-Karte bietet Intel ein umfangreiches Datenblatt an: [Inth].

2.2. Angestrebte Zielgruppe

Dieser Abschnitt beschreibt die Zielgruppe der Xeon-Phi-Karte wie Intel sie selbst einschätzt. Hierzu bietet Intel auf [Gar] eine kurze Übersicht, wann sich der Einsatz der Xeon-Phi-Karte angeblich lohnt oder man doch eher auf konventionelle Hardware zurückgreifen sollte.

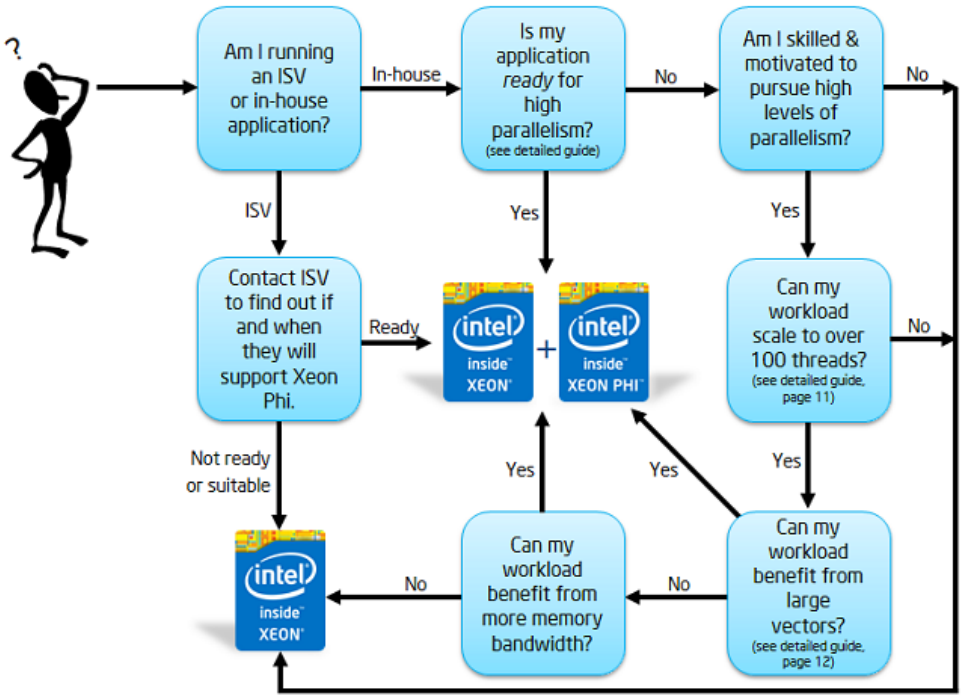


Abbildung 2.3.: Intels Eignungsdiagramm [Gar]

Wie aus dem Diagramm ersichtlich wird, ist das ausschlaggebende Kriterium für eine effiziente Nutzung der Xeon-Phi-Karte die Programmstruktur. Um von den Vorteilen der Xeon-Phi-Karte profitieren zu können, muss das auszuführende Programm hochparallelisierbar sein oder entsprechend umgeschrieben werden können, um die sequentiellen Anteile minimal zu halten. Außerdem sollte das Programm gut skalieren und weiterhin eine entsprechend hohe Leistung erzielen, auch wenn die Thread-Zahl über 100 hinausgeht. Um die große Vektoreinheiten der Kerne auch optimal auszulasten, sollte das Programm Bereiche im Code aufweisen, die vektorisiert werden können - dafür bieten sich

meistens Schleifen an, in deren Rumpf nicht auf gleichen Speicheradressen geschrieben wird. Intel zufolge erfüllen aber nur ca. 15% aller Anwendungen diese Anforderungen, alle anderen Anwendungen könnten auch auf konventionellen Systemen ihre maximale Leistung erreichen. Programme mit großem Umfang sind dabei aber nicht ausgeschlossen, da sie lediglich neu kompiliert und evtl. optimiert werden müssen - sie müssen nicht komplett neu geschrieben werden. [Inta]

Nach Durchführung der Programmanpassungen und Leistungsanalyse in späteren Kapiteln wird das Thema der Zielgruppe im Abschnitt *Tatsächliche Zielgruppe* (Seite 58) nochmal aufgegriffen und unter Berücksichtigung aller Erkenntnisse, die im Laufe der Entstehung der Bachelorarbeit entstanden sind, neu bewertet.

Zusammenfassung

Die Xeon-Phi-Karte bietet eine starke Hardware an, indem jeder Kern mit einer großen Vektoreinheit ausgestattet wurde und die Xeon-Phi-Karte insgesamt bis zu 240 echte Threads zur Verfügung stellen kann. Ein Programm, welches auf der Xeon-Phi-Karte ausgeführt werden soll, muss allerdings einige Voraussetzungen erfüllen, damit sich die Unterstützung des Hosts mit einer Xeon-Phi-Karte auch lohnt, was die Xeon-Phi-Karte für viele Anwendungsbereiche disqualifiziert.

3. Benutzung der Intel Xeon Phi

In diesem Kapitel wird auf die Verwendung der Xeon-Phi-Karte eingegangen, indem die verschiedenen Betriebsmodi, die zusätzlichen Möglichkeiten des Debuggings, welche sich erst durch Verwendung der Xeon-Phi-Karte ergeben bzw. notwendig werden, und weitere Anpassungsmöglichkeiten vorgestellt werden. Dabei handelt es sich aber nur um eine Übersicht ohne Anspruch auf Vollständigkeit, auf die wichtigsten Punkte, die für einen Einstieg in die Thematik notwendig sind, wird aber eingegangen. Für eine umfassendere Beschreibung sei erneut auf die Quellen verwiesen, die bereits in *Beschreibung der Intel Xeon Phi* (Seite 10) erwähnt wurden und die der Autor erneut als Quelle benutzt.

Da *Harmonic Analysis*, womit in Kapitel *Programmanpassungen* (Seite 38) der Offload-Modus demonstriert und auf aufgetretene Probleme eingegangen wird, wie viele andere Programme der Erdsystemforschung auch in Fortran geschrieben worden ist, konzentriert sich dieses Kapitel auf die Konzepte, die auch in Fortran vorhanden sind. Da auf der Xeon-Phi-Karte aber neben Fortran-Programmen auch Programme ausgeführt werden können, die in C/C++ geschrieben worden sind, werden alle wichtigen Code-Bausteine sowohl in Fortran als auch in C/C++ präsentiert.

3.1. Einrichtung

Da die Xeon-Phi-Karte wie ein zusätzlicher Netzwerkknoten behandelt wird, ist die Wahl des Betriebssystems des Hosts relativ unabhängig von der Xeon-Phi-Karte. Nur für die Kommunikation über PCIe mit der Xeon-Phi-Karte müssen für das Betriebssystem auf der Host-Seite die korrekten Treiber bereitgestellt werden. Hierzu wird der Service *MPSS*[1] gestartet, welcher die Xeon-Phi-Karte bootet und in das Netzwerk einbindet, sodass eine Verbindung mittels `ssh` möglich ist [Cog]. Für eine genauere Beschreibung der Einrichtung der Xeon-Phi-Karte sei auf Intels Dokumentation [Rot] und auf [WZS+14, S. 325-331] verwiesen.

Meistens wird die Einrichtung der Xeon-Phi-Karte von einem Systemadministrator übernommen, sodass dieses Thema den Benutzer der Xeon-Phi-Karte nicht betrifft. Man muss allerdings bedenken, dass die unterschiedlichen Betriebsmodi in Abhängigkeit des auszuführenden Programms eine Neukompilierung der verwendeten Bibliotheken

[1] Intel Manycore Platform Software Stack. Offiziell unterstützt MPSS *Red Hat Enterprise* (ab Version 6.0), *SuSE Linux Enterprise Server (SLES)* (ab Version 11) und *Microsoft Windows* (ab Version 7) `https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss` (Zuletzt aufgerufen: 29.11.2014)

erforderlich machen könnten, was möglicherweise in der Verantwortung des Benutzers liegt. Die Anpassungen, die für die Ausführung von *Harmonic Analysis* notwendig waren, werden für den nativen Modus im Abschnitt *Aufgetretene Probleme* (Seite 38) und für den Offload-Modus im Abschnitt *Aufgetretene Probleme* (Seite 42) aufgeführt.

3.2. Betriebsmodi

In dieser Übersicht werden die Betriebsmodi der Xeon-Phi-Karte und ihre Anwendung vorgestellt, die für die meisten Anwendungsprogramme ausreichen. Eine genauere Unterteilung der Betriebsmodi findet man in Kapitel 5 von [WZS+14].

Nativer Modus: Den nativen Modus kann man bezüglich seiner Funktionsweise mit einem Cross-Compiler vergleichen: Der Quellcode wird zwar auf dem Host kompiliert, die Binärdatei liegt danach dann aber in einem Dateiformat vor, welches auf der Xeon-Phi-Karte ausgeführt werden kann (dafür dann aber nicht mehr auf dem Host). Die Quellcode-Anweisungen werden also vollständig auf der Xeon-Phi-Karte ausgeführt.

Offload-Modus: Im Offload-Modus werden Stellen des Quellcodes für den Offload markiert. Wenn der Host bei Programmausführung an diese Stelle gelangt, wird der Aufruf an das Offload-Ziel weitergegeben, welches dann den markierten Bereich ausführt. Nachdem der Bereich vollständig ausgeführt wurde, kehrt der Aufruf zum Host zurück und führt den Code nach dem markierten Bereich weiter aus.

Symmetrischer Modus: In diesem Modus wird vom nativen Modus oder Offload-Modus Gebrauch gemacht. Zusätzlich wird der Nachrichtenaustausch mittels MPI genutzt, sodass Host und das Offload-Ziel parallel zueinander weiterarbeiten können

3.2.1. Nativer Modus

Bezüglich der Anwendbarkeit ist der native Modus der einfachste Bedienmodus der Xeon-Phi-Karte. Im Optimalfall kann ein Progamm, welches in C/C++/Fortran geschrieben worden ist, unverändert auf einer Xeon-Phi-Karte ausgeführt werden. Die einzige notwendige Änderung ist hierbei eine Neukompilierung des Programmes mit zusätzlichem `-mmic`-Compilerflag. Wenn der Compiler dies unterstützt, wird er die ausführbare Binärdatei in das Dateiform `ELF 64-bit LSB executable, Intel Xeon Phi coprocessor (k1om)` kompilieren. Die Binärdatei kann dann nicht mehr auf dem Host sondern nur noch auf einer Xeon-Phi-Karte ausgeführt werden. Der Benutzer muss die Binärdatei dann nur noch über das Netzwerk auf die Xeon-Phi-Karte rüberkopieren, sich dann direkt auf der Xeon-Phi-Karte mittels `ssh` einloggen und das Programme dann starten. Das Programm läuft dann vollständig auf der Xeon-Phi-Karte und der Host trägt nichts zur Rechenleistung mit bei.

Da die Xeon-Phi-Karte in diesem Modus aber nicht mit dem Host interagiert, müssen neben dem Binärfile auch alle verwendeten Daten auf die Xeon-Phi-Karte kopiert werden. Das betrifft sowohl Dateien zum Lesen/Schreiben während der Programmausführung als auch verwendete Bibliotheken. Damit die Bibliotheken aber auch auf der Xeon-Phi-Karte eingebunden werden können, müssen diese vor dem Kopieren auf dem Host wie die Binärdatei mit dem `-mmic`-Flag neu kompiliert werden. Der Speicherort der rüberkopierten Bibliotheken kann auf der Xeon-Phi-Karte mit `export LD_LIBRARY_PATH=pfadZurBib` festgelegt werden. Einige wichtige Bibliotheken (wie z.B. die OpenMP-Bibliothek `libiomp5.so`) liefert Intel bereits fertig kompiliert im Ordner */opt/intel/composerxe/lib/-mic/* mit. Tabelle 3.1 bietet eine Übersicht der wichtigsten vorkompilierten Bibliotheken, die Intel für den nativen Modus mitliefert. Außerdem stehen die GNU C Bibliotheken bereits auf der Xeon-Phi-Karte zur Verfügung und müssen nicht rüberkopiert werden.

Bibliothek	Beschreibung
libcilkrts.so.5	Intel Cilk Plus runtime library
libifcoremt.so.5	Thread-safe Intel-specific Fortran run-time library
libifport.so.5	Portability and POSIX support
libimf.so	Math library
libintlc.so.5	Intel support libraries for CPU dispatch, intel_fast_*, and traceback support routines
libiomp5.so	Compatibility OpenMP* dynamic runtime library
libsvml.so	Short vector math library
libirng.so	Random number generator library
libmpi.so.4.0, libmpigf.so.4.0, libmpigc4.so.4.0	Intel® MPI runtime libraries
libicaf.so	Coarray Fortran library

Tabelle 3.1.: Intels vorkompilierte Bibliotheken [Amaa]

Um zu erfahren, was für Abhängigkeiten eine Binärdatei aufweist, damit man die benötigten Bibliotheken neu kompilieren und auf die Xeon-Phi-Karte kopieren kann, kann man das Tool `micnativeloadex` nutzen. Hierzu muss man zunächst in der Umgebungsvariablen `SINK_LD_LIBRARY_PATH` die Pfade zu den Laufzeitbibliotheken für die Xeon-Phi-Karte und der Binärdatei setzen. Der Aufruf des Tools zeigt dann alle Abhängigkeiten der Binärdatei an, die man berücksichtigen muss [Amaa].

```
export SINK_LD_LIBRARY_PATH =
   ↪ /opt/intel/composer_xe_2013_sp1.X.XXX/compiler/lib/mic/
   ↪ :pfadZumBinary
/opt/intel/mic/bin/micnativeloadex binaerDatei -l
```

Listing 3.1: Dynamische Abhängigkeiten anzeigen

3.2.2. Offload

Wie im Abschnitt *Zusammenspiel der Bauelemente* (Seite 11) gezeigt wurde, sind Host und Xeon-Phi-Karte über PCI Express miteinander verbunden, der Host kann aber nicht selbstständig auf die Xeon-Phi-Karte zugreifen, um durch eigenständige Nutzung der Xeon-Phi-Karte seine Rechenleistung zu erhöhen[Kin]. Er kann aber mittels Offload die Rechenkapazitäten der Xeon-Phi-Karte mitverwenden, indem Teile des Programms zur Ausführung auf die Xeon-Phi-Karte ausgelagert werden. Im Offload-Modus wird der Quellcode an speziell markierten Stellen über das Netzwerk aufgeteilt. Die Markierungen werden mit Direktiven gesetzt, die nicht erst zur Laufzeit sondern bereits beim Kompilieren ausgewertet werden. Dieses Vorgehen ist mit den Compiler-Direktiven aus OpenMP vergleichbar und hat ähnliche Vorteile: Wenn ein Compiler die OpenMP-Direktiven nicht interpretieren kann, werden sie ignoriert und statt einen Fehler zu werfen, wird eine Binärdatei ohne Parallelcode erzeugt. Auf den Offload-Modus bezogen passiert ähnliches: Vorausgesetzt der Compiler kann mit den Offload-Direktiven umgehen, erstellt er dann mehrere Binärdateien.

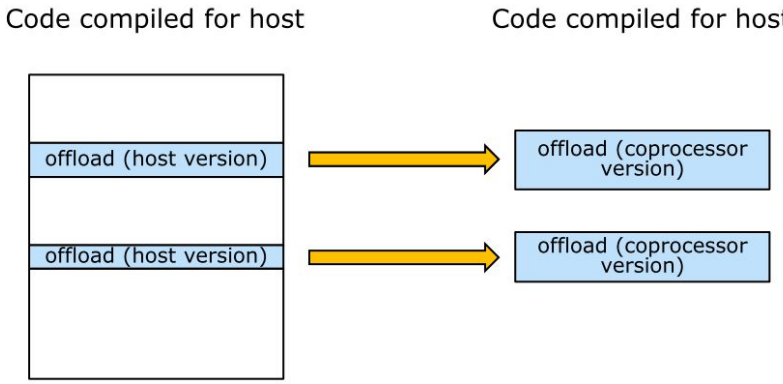


Abbildung 3.1.: Binärdateiverteilung [Intd, S.13]

Die Binärdatei für den Host enthält den kompletten Code inklusive der markierten Bereiche, die Offload-Ziele erhalten dann eine reduzierte Binärdatei, in denen nur der Code enthalten ist, der auch tatsächlich von ihnen ausgeführt werden soll (vgl. Abbildung 3.1). Wenn die Host-Binärdatei dann ausgeführt wird und zum Offload-Code gelangt, prüft die Laufzeitumgebung, ob das Offload-Ziel auch erreichbar ist. Sollte dies nicht der Fall sein, kann das Programm trotzdem vollständig auf dem Host ausgeführt werden. Andernfalls lädt die Laufzeitumgebung die reduzierte Binärdatei, und alle benötigten Bibliotheken und Variablenbelegungen auf das Offload-Ziel und führt den Offload-Code aus. Variablenbelegungen, die im Offload-Bereich verwendet werden, werden per default kopiert und auf das Ziel geladen. Bei Rückkehr des Aufrufs wird der Wert dann zurückkopiert. Dieses Verhalten kann durch Anweisungen aber auch geändert werden. Dieses Konzept wird *explizites Speichermodell* genannt - es gibt außerdem noch ein *implizites Speichermodell*, in dem für den Host und das Offload-Ziel geteilter Speicher simuliert wird.

Dieses Modell ist allerdings nur für C/C++ und nicht für Fortran verfügbar, weshalb hier nicht weiter darauf eingegangen werden soll.

Außer um die Markierungen muss sich der Benutzer dabei aber um nichts kümmern, da die Laufzeitumgebung die korrekte Binärdateiverteilung abwickelt. Es ist aber auch möglich, das Offload-Ziel explizit anzugeben, wenn z.B. Code auf eine bestimmte Xeon-Phi-Karte unter vielen anderen gesendet werden soll. Sollte das explizit angegebene Ziel dann aber zur Laufzeit nicht erreichbar sein, wird ein Fehler geworfen. Offload-Ziel und Host sind hierbei allgemein zu verstehen, das Offload-Ziel muss nicht unbedingt auch eine Xeon-Phi-Karte sein. Das ist vor allem im symmetrischen Modus von Bedeutung. Im weiteren Verlauf wird die Xeon-Phi-Karte aber als Offload-Ziel betrachtet.

Syntax

```
!dir$ offload begin <clauses>
...
!dir$ end offload
```

Listing 3.2: Offload-Syntax (Fortran)

```
#pragma offload_attribute(push, target(mic))
...
#pragma offload_attribute(pop)
```

Listing 3.3: Offload-Syntax (C/C++)

In Listing 3.2 und 3.3 sind die Compiler-Direktiven angegeben, die Code-Bereiche zum Offload freigeben. Wichtige Clauses und Modifier sind:

Clause	Erklärung
`target(mic)`	Der Code wird auf einer beliebigen, verfügbaren Xeon-Phi-Karte ausgeführt. Ist keine Verfügbar, führt der Host den Code aus
`target(n)`	Der Code wird auf der Xeon-Phi-Karte mit der Nummer $n+1$ ausgeführt
`target(-1)`	Wie `target(mic)`, ist aber keine Xeon-Phi-Karte verfügbar, wird ein Laufzeitfehler geworfen
`in(var-list modifiers)`	`var-list` wird von Host auf Xeon-Phi-Karte kopiert
`out(...)`	`var-list` wird von Xeon-Phi-Karte auf Host kopiert
`inout(...)`	`var-list` wird von Host auf Xeon-Phi-Karte und wieder zurück kopiert
`nocopy(...)`	kein Kopiervorgang, `var-list` ist nur lokal auf der Xeon-Phi-Karte verfügbar

Tabelle 3.2.: Wichtige Clauses

Modifier	Erklärung
`alloc_if(bool)`	Allokiere Speicher auf der Xeon-Phi-Karte
`free_if(bool)`	Deallokiere Speicher auf der Xeon-Phi-Karte

Tabelle 3.3.: Wichtige Modifier

Dies sind die wichtigsten Offload-Direktiven, Clauses und Modifier, die für viele Zwecke aber auch bereits ausreichend sind, eine erweiterte Übersicht gibt es auf [BBI+, Kapitel 4.3].

Die Modifier `alloc_if(bool)` und `free_if(bool)` sind besonders dann interessant, wenn ein Datenverbund in verschiedenen Offload-Blöcken vorkommt. Somit kann man sich nämlich z.B. eine Deallokierung am Ende des ersten und eine Allokierung am Anfang des zweiten Offload-Blockes sparen, wenn dieser Datenverbund darin verwendet wird. Wenn man einen Datenverbund so durch mehrere Offload-Blöcke durchreicht und der Host diesen nicht benötigt, kann man zudem Kopiervorgänge einsparen, indem man diese mit `nocopy(var-list modifiers)` unterbindet.

Bei der Benutzung des Offload-Modus ist außerdem noch zu beachten, dass im Offload-Modus alle relevanten Umgebungsvariablen des Hosts auf die Xeon-Phi-Karte kopiert werden. Das ist aber nicht immer erwünscht, da z.B. auch `OMP_NUM_THREADS` kopiert wird und die optimale Zahl benutzter Threads auf der Xeon-Phi-Karte und dem Host vermutlich unterschiedlich sind. Um das zu verhindern, kann man mit `MIC_ENV_PREFIX` ein Präfix definieren, sodass nur Umgebungsvariablen mit diesem Präfix hochgeladen werden. Wenn man dieses Präfix setzt, muss man es für alle Umgebungsvariablen setzen,

die auf der Xeon-Phi-Karte verwendet werden sollen. Ein Beispiel wird in Listing 3.4 gezeigt, wodurch auf dem Host und der Xeon-Phi-Karte unterschiedlich viele Threads benutzt werden. Der Host benutzt dann den Wert aus `OMP_NUM_THREADS` und die Xeon-Phi-Karte den Wert aus `M_OMP_NUM_THREADS`. [Intd]

```
OMP_NUM_THREADS=8
MIC_ENV_PREFIX=M_
M_OMP_NUM_THREADS=124
```

Listing 3.4: Beispiel für unterschiedliche Umgebungsvariablenbelegung [Intb]

3.2.3. Symmetrischer Modus

Der symmetrische Modus ermöglicht es, dass der Host und die Xeon-Phi-Karte gleichzeitig arbeiten, wohingegen im nativen Modus nur die Xeon-Phi-Karte und im Offload-Modus die Xeon-Phi-Karte und der Host abwechselnd aber nicht gleichzeitig arbeiten. In diesem Modus werden alle Xeon-Phi-Karten und der Host als ein Netzwerk mit verteiltem Speicher betrachtet und kommunizieren mittels MPI miteinander. Dafür muss man dann auch auf den nativen Modus oder den Offload-Modus zurückgreifen. Nutzt man MPI zusammen mit dem Offload-Modus, kann man den MPI-Rängen eigene Offload-Blöcke zuweisen, die dann parallel ausgeführt werden. Alternativ kann man auch unterschiedliche Quellcode-Versionen erstellen, diese im nativen Modus kompilieren und auf den Host und die Xeon-Phi-Karten kopieren - der gleichzeitige Programmstart erfolgt dann mit folgendem MPI-Aufruf:

```
export I_MPI_MIC=enable
mpirun -n <# of processes> -host <hostname1> <application>
   ↪ : -n <# of processes> -host <hostname2> <application>
   ↪ : ...
```

Listing 3.5: MPI-Programmstart mit nativen Programmversionen

Die Umgebungsvariable `I_MPI_MIC` muss gesetzt werden, um die Kommunikation zwischen Host und Xeon-Phi-Karten mittels MPI zu erlauben. Bei der Programmausführung ist nicht notwendigerweise der Host involviert, auch die Verteilung der Prozesse muss nicht gleichmäßig sein. Da dieser Modus nicht zur Anwendung für *Harmonic Analysis* kam, bleibt es bei dieser oberflächlichen Beschreibung. Ein Anwendungsbeispiel und weitere Erklärungen findet man auf [Intj] und [BBI+, Kapitel 6.2].

3.3. Auswahl des Betriebsmodus

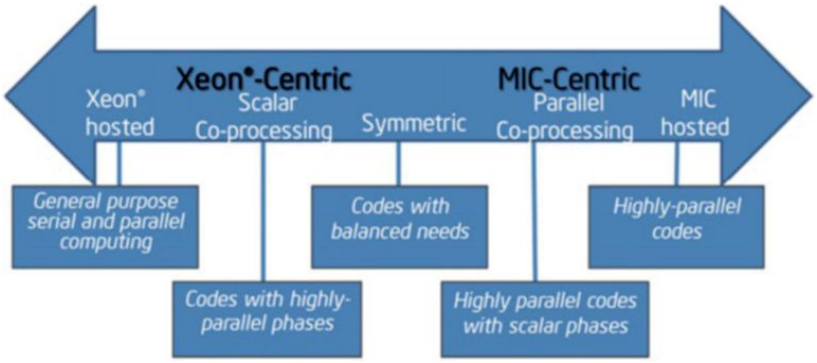


Abbildung 3.2.: Wahl des Betriebsmodus [WZS+14, S.75]

Abbildung 3.2 bietet eine Übersicht, für welche Programmarten sich welcher Modus anbietet. Bezüglich des Arbeitsaufwandes wird davon ausgegangen, dass alle benötigten Bibliotheken bereits im richtigen Format vorliegen. Je nach Gestalt des Programmes, welches mit der Xeon-Phi-Karte ausgeführt werden soll, erhält man unterschiedliche Empfehlung zur Wahl des Betriebsmodus.

- **Xeon hosted:** Programme, die sich nicht so recht in die anderen Kategorien einordnen lassen und sich nicht durch hochparallelisierbare Programmabschnitte auszeichnen, sollten auf die Verwendung der Xeon-Phi-Karte verzichten und vollständig auf dem Host ausgeführt werden (vgl. auch Abbildung 2.3, Seite 13). Hierbei ist dann auch kein Zusatzaufwand notwendig.

- **Scalar Co-processing:** Wenn der Code Stellen aufweist, die sehr gut parallelisiert werden können, ansonsten aber sequentiell abläuft oder nur bedingt mit hoher Threadzahl skaliert, sollte das Programm im Offload-Modus ausgeführt werden. So können die ungeeigneten Abschnitte auf dem Host bearbeitet werden, um dann den sehr gut skalierenden Code auf der Xeon-Phi-Karte auszuführen. Der Arbeitsaufwand hierbei ist überschaubar, da nur Compiler-Direktiven eingebaut werden müssen.

- **Symmetric** und **Parallel Co-processing:** Die Xeon-Phi-Karte und der Host arbeiten parallel auf angepassten Programmversionen, welche sich durch viele Abschnitte mit sehr gut skalierendem Code und vektorisierbaren Schleifen auszeichnet. Hierzu wird eine Kombination aus MPI und Offload- bzw. nativem Modus verwendet, was dann zu einem hohen Arbeitsaufwand führt.

- **MIC hosted:** Das Programm wird im nativen Modus allein auf der Xeon-Phi-Karte ausgeführt. Dieser Modus ist nur dann für ein Programm geeignet, wenn es hauptsächlich Code beinhaltet, der sehr gut mit der Threadzahl skaliert, und wenig I/O aufweist.

3.4. Weitere Anpassungsmöglichkeiten

Neben der Verwendung der unterschiedlichen Betriebsmodi kann man noch weitere Anpassungen am Programm vornehmen, vier Möglichkeiten werden hier vorgestellt [Inte]:

Verwendung optimierter Bibliotheken: Intel stellt eine eigene Mathematik-Bibliothek, MKL[2], zur Verfügung, die viele verschiedene Funktionen enthält, unter anderem die aus LAPACK[3] und BLAS[4]. Alle Funktionen sind für das Hochleistungsrechnen optimiert und werden in Abhängigkeit der erkannten Hardware ausgewählt. Nutzt das zu optimierende Programm Funktionen, die auch in MKL enthalten sind, könnte ein Wechsel auf diese die Programmlaufzeit verbessern.

Variation der Threadzahl: Jeder Kern der Xeon-Phi-Karte unterstützt maximal vier echte Threads. Das bedeutet aber nicht, dass jedes Programm mit maximaler Threadzahl auch die maximal erreichbare Leistung erreicht. Dies hängt aber nicht nur mit der Skalierbarkeit des Programmes zusammen, sondern auch mit der Hardware der Xeon-Phi-Karte. Da jeder Kern der Xeon-Phi-Karte nur eine Vektoreinheit hat, kommt es bei zu vielen Threads, die gleichzeitig Zugang zur Vektoreinheit benötigen, zu Verzögerungen, bei zu wenigen ist die Auslastung der Vektoreinheit nicht optimal. Ebenso kann es auch zu viele oder wenige Zugriffe auf den L1-Cache und L2-Cache geben. Den optimalen Wert findet man nur durch Laufzeittests heraus - es hat sich aber gezeigt, dass für viele Anwendungen 3 Threads pro Kern optimal sind.

Thread-Verteilung: Das Setzen der Umgebungsvariable `KMP_AFFINITY` auf `compact`, `balanced` oder `scatter` gibt an, wie die Threads auf den Kernen verteilt werden. Je nachdem wie das zu parallelisierende Programm aufgebaut ist, kann es dabei von Vorteil sein, bei einer Threadzahl unter dem Maximum diese gleichmäßig auf allen Kernen zu verteilen oder einige Kerne voll aufzufüllen und die restlichen leer zu belassen. Bei der gleichmäßigen Verteilung kann außerdem die Art der Verteilung eingestellt werden. Wie schon bei der Variation der Threadzahl findet man die optimale Verteilung am leichtesten mittels Laufzeittests mit allen Varianten, Abbildung 3.3 (Seite 24) zeigt eine grafische Darstellung dieser Varianten, in der acht Threads auf vier Kerne aufgeteilt werden.

Vektorisierung: Der Compiler automatisiert ab Optimierungslevel `-O2` selbstständig Schleifen, wenn sie zwei Voraussetzungen erfüllen:

1. Korrektheit: Eine Vektorisierung der Schleife verändert nicht das Endergebnis. Dies ist z.B. dann nicht gewährleistet, wenn in einem Schleifendurchgang auf gleiche Speicheradressen lesend und schreibend zugegriffen wird (Race Condition).

[2] Math Kernel Library, eine Übersicht enthaltener Funktionen und Information zur Anwendung findet man in [WZS+14, Kapitel 7]
[3] Linear Algebra Package
[4] Basic Linear Algebra Subprograms

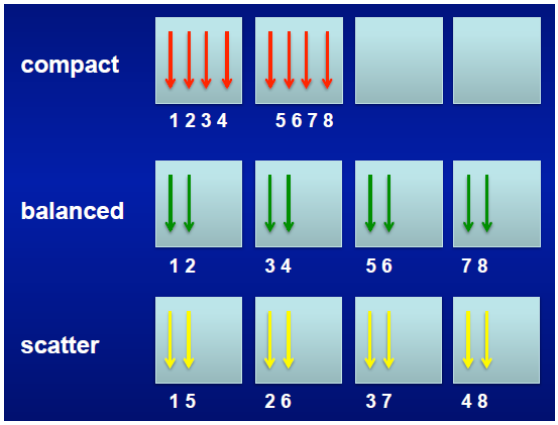


Abbildung 3.3.: Threadverteilungsmethoden [Wil, S.12]

2. Nutzen: Eine Vektorisierung hat das Potential die Laufzeit tatsächlich zu verbessern

Nur wenn der Compiler beide Voraussetzungen als garantiert erfüllt erkennt, wird die Schleife automatisch vektorisiert. Sollte der Compiler auch nur bei einer Voraussetzung keine Garantie für deren Erfüllung aussprechen können, wird er nicht vektorisieren. Der Benutzer kann die Durchführung der Vektorisierung aber auch beeinflussen, indem er die entsprechenden Compiler-Direktiven vor die Schleife setzt. Die wichtigsten Direktiven sind [Bla]:

Fortran	C/C++	Beschreibung
!dir$ vector	#pragma vector always	Der Compiler geht davon aus, dass der Nutzen garantiert ist
!dir$ ivdep	#pragma ivdep	Wenn der Compiler die Korrektheit nicht sicher widerlegen kann, geht er davon aus, dass die Korrektheit garantiert ist
!dir$ simd	#pragma simd	Die Vektorisierung durch den Compiler wird erzwungen (sollte mit Vorsicht genutzt werden, da es zu vorschneller Handlung verleitet)
!dir$ novector	#pragma novector	Der Compiler führt keine Vektorisierung durch

Tabelle 3.4.: Vektorisierungs-Direktiven

3.5. Debugging

In diesem Abschnitt werden einige Möglichkeiten vorgestellt, den Effekt der Programmanpassungen zu verfolgen, um so gegebenenfalls fehlerhaftes Verhalten aufdecken zu können.

Häufig wird man anfangen, die Bestandteile eines Programms, in denen man das größte Potential für eine parallele Ausführung und Vektorisierung vermutet, mittels Offload-Anweisungen auf der Xeon-Phi-Karte auszuführen. Mithilfe des Makros `__MIC__`, welches nur auf der Xeon-Phi-Karte definiert ist, kann getestet werden, ob Quellcode auf dem Host oder der Xeon-Phi-Karte ausgeführt wird.

```
#ifdef __MIC__
    write(*,*) "Ich bin auf der Xeon PHi"
#else
    write(*,*) "Ich bin nicht auf der Xeon Phi"
#endif
```

Listing 3.6: Verwendung von ___MIC___

Desweiteren gibt es einige Compiler-Flags, die zusätzliche Informationen liefern:

-vec-report=X Beim Kompilieren werden je nach Wahl von X unterschiedliche Informationen über den Status der Vektorisierung aller Schleifen des Programmes angezeigt. Es gibt zwar noch eine Stufe X=7, die ausgegebenen Informationen sind aber ohne die Auswertung mit Skripten nicht mehr nachvollziehbar.

- X=0: Keine zusätzlichen Bildschirmausgaben
- X=1: Anzeige von Schleifen, die vektorisiert wurden
- X=2: wie X=1 und Anzeige von nicht vektorisierten Schleifen mit Begründung
- X=3: wie X=2 mit zusätzliche Informationen (z.B. vermutete Abhängigkeiten)
- X=4: Anzeige von nicht-vektorisierten Schleifen
- X=5: wie X=4 mit zusätzlichen Informationen
- X=6: wie X=3 mit noch mehr Informationen

-guide-vec Beim Kompilieren wird für jede innere Schleife angezeigt, was der Benutzer tun kann, damit diese vektorisiert wird

-opt-report Beim Kompilieren werden die Optimierungsvorgänge seitens des Compilers angezeigt

-no-offload Compiler-Direktiven für den Offload-Modus werden ignoriert, ermöglicht schnelles Umschalten zwischen dem „normalen“ Modus und Offload-Modus

Außerdem kann man sich zur Programmlaufzeit Informationen zum Offload anzeigen lassen. Hierzu muss man nur die Umgebungsvariable `OFFLOAD_REPORT` auf den Wert 1 oder 2 setzen. An den Stellen, wo Quellcode mittels Offload auf die Xeon-Phi-Karte geladen und ausgeführt wird, wird mit `OFFLOAD_REPORT=1` die Zeile des Offload-Aufrufes sowie Laufzeiten angezeigt, mit `OFFLOAD_REPORT=2` werden dann zusätzlich noch die übertragenen Datenmengen aufgeführt. Somit erhält man einen Eindruck, an welchen Stellen wie viel kopiert oder gerechnet wird. Da der Offload-Report die Leistung insgesamt aber deutlich bremst, muss man für eine Leistungsanalyse wieder auf bewährte Mittel zurückgreifen.

```
export OFFLOAD_REPORT=2
./job_mpiom_tides_kleinerDatensatz.sh 1000
[...]
process =            0  ha_terms_get_4d_netcdf :: Got data
   ↪ = zo from:
          1  to:          1000  time [sec]:    2.881975
[Offload] [MIC 0] [File]
   ↪ /home/dkrz/k203089/petra/projects/fortran-utils/
   ↪ modules/harmonic_analysis/
   ↪ mod_harmonic_analysis_prepare.f90
[Offload] [MIC 0] [Line]             305
[Offload] [MIC 0] [Tag]              Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]         13.675269
   ↪ (seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]    297471680
   ↪ (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]         5.417083
   ↪ (seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]    19183176 (bytes)

 process =             0  threads =           180
    ↪ ha_terms_get_4d_netcdf
[...]
```

Listing 3.7: Beispiel für `OFFLOAD_REPORT`

Listing 3.7 zeigt ein Beispiel des Offload-Reports anhand von *Harmonic Analysis*, welches mit dem kleinen Datensatz, 180 Threads und $Chunksize = 1000$ ausgeführt wurde. Im Offload-Report kann man u.a. erkennen, dass der Offload in Zeile 305 in *mod_harmonic_-analysis_prepare.f90* insgesamt knapp 280MB an Daten auf die Xeon-Phi-Karte kopiert, der Rechenvorgang auf der Karte insgesamt 5,4 Sekunden dauert und am Ende ca. 18MB wieder zurück auf den Host kopiert werden.

3.6. Weitere Arbeiten

Eine frühe Analyse der Performance einer Intel Xeon Phi SE10P, die bis auf einen Kern mehr und eine leicht erhöhte Taktrate der hier untersuchten Intel Xeon Phi 5110P ähnlich ist, findet man in diesem Paper: [SKC13]. Die Autoren dieses Papers untersuchen die Performance der Xeon Phi, indem sie auf der Karte das Produkt einer dünnbesetzten Matrix mit einem Vektor berechnen lassen.

Ein weiteres, interessantes Paper ist [FSZ+], in dem die Autoren die Leistung einer Intel Xeon Phi der 5100-Serie untersuchen. Hierzu verwenden sie in einem ersten Schritt verschiedene Benchmarks, um die unterschiedlichen Komponenten der Xeon-Phi-Karte zu testen. Zum Abschluss nutzen sie die Xeon-Phi-Karte dann, um eine Anwendung aus dem medizinischen Bereich zur Bildanalyse von Mikroskopaufzeichnungen auszuführen, um so die Einsatzfähigkeit der Xeon-Phi-Karte für reale Anwendungen zu untersuchen.

Zusammenfassung

Wenn die Karte erstmal eingerichtet worden ist, können Programme auf drei unterschiedliche Weisen auf der Xeon-Phi-Karte ausgeführt werden: das Programm läuft im nativen Modus vollständig auf der Xeon-Phi-Karte, das Programm läuft abwechselnd mittels Offload auf der Xeon-Phi-Karte und dem Host oder das Programm wird im symmetrischen Modus parallel auf der Xeon-Phi-Karte und dem Host ausgeführt. Vorausgesetzt alle benötigten Bibliotheken wurden korrekt kompiliert, ist es am einfachsten, den nativen Modus zu verwenden, was aber nur in speziellen Fällen bereits eine optimale Leistung einbringt. Der Offload-Modus hingegen wird mit geringem Arbeitsaufwand bereits gute Leistungsergebnisse einbringen. Der symmetrische Modus ist zwar mit viel Arbeit verbunden, um das Programm anzupassen, hat aber das Potential, die Laufzeit weiter zu optimieren. Außerdem sollte auch ein besonderes Augenmerk auf die Vektorisierung gelegt werden, um die Fähigkeiten der Xeon-Phi-Karte voll auszunutzen.

4. Programmbeschreibung

Das Verständnis der Gezeiten und ihre Vorhersage mittels Gezeitentafeln ist eine Wissenschaft, die von großer praktischer Bedeutung für Küstenbewohner und die Seefahrt ist. Bereits mehrere Jahrhunderte vor unserer Zeit erkannten Menschen, dass es einen Zusammenhang zwischen dem Stand des Mondes und den auftretenden Gezeiten gab [Wika]: Ein Mondtag, d.h. die Zeit bis der Mond unter Berücksichtigung der Erdrotation und seines eigenen Umlaufs erneut über dem gleichen Längengrad steht, dauert etwa 24 Stunden und 50 Minuten und ähnelt somit an vielen Weltmeeresküsten der zeitlichen Verschiebung der Gezeiten. Seit dem 19. Jahrhundert bediente man sich zusätzlich des Ansatzes, dass die Gezeiten einer erzwungenen Schwingungsperiode folgen, welche als Serie harmonischer Schwingungen dargestellt werden kann [Wikb]. Diese erzwungenen Schwingungsperioden folgen ihren natürlichen Ursachen: In erster Linie ist dies der Einfluss der Mondegravitation, gefolgt von der Sonnengravitation. Durch die relative Bewegung von Sonne und Mond ergeben sich außerdem verschiedene Schwingungen unterschiedlicher Stärke und Periode für die Himmelskörper. Hinzu kommen auch nicht-astronomische Einflüsse wie z.B. Winde, Topographie, Meeresströmungen, Corioliskraft usw. Je nachdem wie stark diese nicht-astronomischen Einflüsse sind, müssen für unterschiedliche Orte unterschiedlich viele Schwingungen berücksichtigt werden, um die gleiche Qualität bezüglich der Gezeitenvorhersage zu erreichen. Für eine ausführlichere Beschreibung der Gezeitenentstehung sei auf [DKKS75, Kapitel 9] verwiesen.

Mittels einer harmonischen Analyse und eines ausreichend großen - bezüglich der Zeitspanne und des Samplings, in der die Daten gemessen wurden - Datensatzes können die einzelnen Schwingungen, deren Periode z.B. durch Beobachtung der Bewegung der Himmelskörper bekannt ist, herausgerechnet werden. *Harmonic Analysis* ist ein Programm, welches eine solche harmonische Analyse auf Gezeitendaten ausführt und die voreingestellten Partialtiden berechnet und speichert. Geschrieben wurde es von Petra Nerge, Mitarbeiterin des Arbeitsbereiches *Wissenschaftliches Rechnen* an der Universität Hamburg.

4.1. Verwendungszweck

Der Hintergrund und die Durchführung der harmonischen Analyse wird noch im Abschnitt *Hintergrund der harmonischen Analyse* (Seite 31) beschrieben. Als Eingabedaten werden für diese Bachelorarbeit Daten genommen, die von Petra Nerge mit *MPIOM* (vgl. Abschnitt *Übersicht*, Seite 8) berechnet worden sind, das Programm kann aber auch real gemessene Gezeitendaten verarbeiten.

4.1.1. Input

Vereinfacht gesagt berechnet *MPIOM* über einen bestimmten Zeitraum für die gesamte Erdoberfläche die Gezeitenstände. Für die Abbildung der Erde wird ein 362×192-Gitter verwendet, die Tiefe des Meeres wird vernachlässigt und auf 1 gesetzt - in der Hinsicht wird einzig zwischen Land und Meer unterschieden. Für jeden Gitterpunkt werden dann in zeitlichen Abständen die jeweiligen Gezeitenstände gespeichert. Die Eingabe-Daten liegen im NetCDF[1]-Format vor - NetCDF ist ein binäres Dateiformat, welches gegenüber der Speicherung von Daten als Textdatei unter anderem die Vorteile besitzt, dass es durch die Speicherung von Metadaten selbstbeschreibend ist, parallele I/O-Operationen erlaubt und Datenkompression ermöglicht [Net]. Im nachfolgenden Listing ist ein Auszug aus den Metadaten abgebildet, in dem die Beschreibung der vier Variablen abgelegt ist, die für die harmonische Analyse benötigt werden.

```
netcdf geogem0032_tidal_lsa01_mpiom_data_zo_ts_2004-12-01_
   ↪ 2004-12-31 {
dimensions:
        x = 362 ;
        y = 192 ;
        depth = 1 ;
        time = UNLIMITED ; // (1488 currently)
variables:
        double lon(y, x) ;
                lon:standard_name = "longitude" ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:_CoordinateAxisType = "Lon" ;
        double lat(y, x) ;
                lat:standard_name = "latitude" ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:_CoordinateAxisType = "Lat" ;
        double time(time) ;
                time:standard_name = "time" ;
                time:units = "hours since 2004-12-01
                   ↪ 00:00:00" ;
                time:calendar = "365_day" ;
        float zo(time, depth, y, x) ;
                zo:standard_name = "sea_surface_elevation" ;
                zo:long_name = "Sea surface elevation" ;
                zo:units = "m" ;
                zo:code = 1 ;
```

[1]Network Common Data Form

```
        zo:coordinates = "lon lat" ;
        zo:_FillValue = -9.e+33f ;
```

Listing 4.1: NetCDF-Header(Auszug)

Harmonic Analysis wurde mit zwei verschiedenen Datensätzen im NetCDF-Format ausgeführt, die über unterschiedliche Zeiträume erstellt wurden. Der kleine Datensatz enthält die Gezeitenstände für alle Gitterpunkte über den Zeitraum eines Monats und hat eine Größe von ca. 792 MB. Der große Datensatz enthält ebenfalls die Gezeitenstände für alle Gitterpunkte, der Messzeitraum geht hier aber über 12 Monate - seine Größe beträgt etwa 9,1 GB. Für beide Datensätze gilt, dass das Messzeitintervall eine halbe Stunde beträgt. Die verwendeten Datensätze enthalten zudem noch weitere Werte für mehrere andere Variablen, die aber nicht alle für die Durchführung der harmonischen Analyse benötigt werden und daher ignoriert werden.

In *Harmonic Analysis* werden die relevanten Daten dann in einem vierdimensionalen Array *inData* gespeichert: in den ersten beiden Dimensionen, werden die x- und y-Koordinaten gespeichert, in der dritten Dimension der Gezeitenstand und in der letzten Dimension der jeweilige Zeitschritt.

4.1.2. Output

Harmonic Analysis schreibt die Ergebnisse der harmonischen Analyse in eine Datei im NetCDF-Format zurück. Zu den Ergebnissen gehören die Amplituden und Phasen der Partialtiden, deren Frequenz beim Programmstart in einer Konfigurationsdatei übergeben worden sind. In der derzeitigen Einstellung werden diese Werte für folgende Partialtiden ausgegeben:

$$2N2, K1, K2, M2, M4, Mf, Mm, N2, O1, P1, Q1, S1, S2$$

Die Bedeutung und Frequenzen dieser Partialtiden kann im Anhang B (Seite 73) eingesehen werden. Mittels `ncview` kann auch eine grafische Darstellung der Amplitude bzw. Phase angezeigt werden. Für die Partialtide $M2$ ergibt sich z.B. folgende Darstellung:

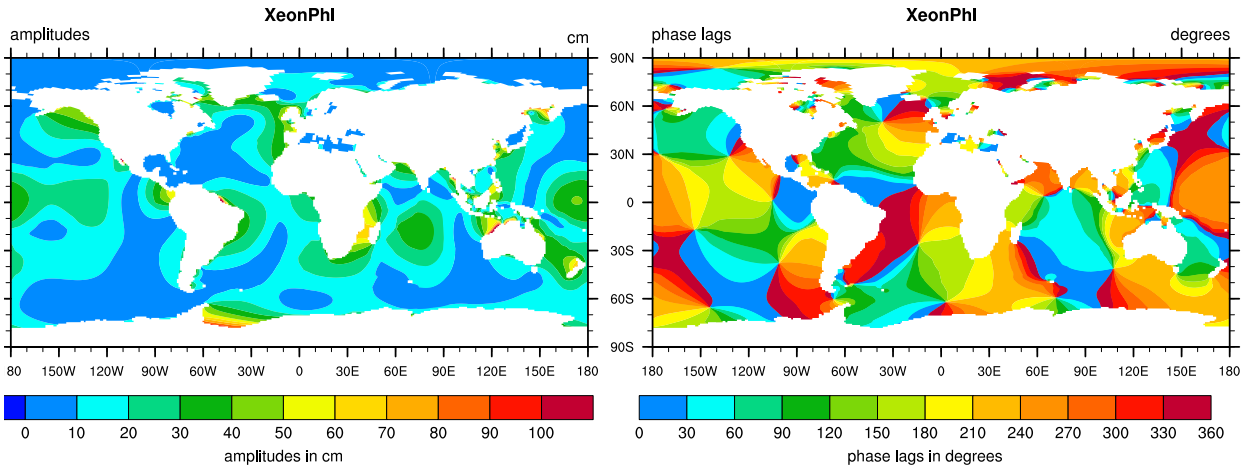


Abbildung 4.1.: Amplitude und Phase der M2-Partialtide

4.2. Hintergrund der harmonischen Analyse

Die harmonische Analyse ist eine sogenannte Zeitreihenanalyse und ist eine Variation der Fourier-Analyse. Diese Analyse wird im aktuell betrachteten Anwendungsfeld (Gezeiten) auf Datenpunkten ausgeführt, die z.B. mittels Sensorbojen, Schiffen oder Satelliten in zeitlichen Abständen gesammelt werden und somit nur diskret anfallen - kontinuierliche Messungen können ansonsten auch leicht durch Sampling in diskrete Datenpunkte umgewandelt werden. Das Ziel der Zeitreihenanalyse ist es, das Eingabesignal - dargestellt durch die Datenpunkte - in seine einzelnen Bestandteile aufzuspalten, deren Zusammenkommen das gemessene Signal ergeben. Die Datenpunkte werden also als die Linearkombination von periodischen Funktionen aufgefasst.

Die Zeitreihenanalyse ermöglicht es also, aus der Wirkung, in Form gemessener Daten, mögliche Ursachen und ihre Stärke abzuleiten, deren Kombination die gemessenen Datenpunkte approximiert. Sobald die Signalzusammensetzung bekannt ist, können dann weitergehende Untersuchungen angestellt werden: z.B. könnten so relevante Ursachen vom Hintergrundrauschen getrennt oder Prognosen über die zukünftige Entwicklung des Datensignals angestellt werden. Sind die Frequenzen der Bestandteile bereits bekannt, kann die Analyse auch noch in Form der harmonischen Analyse weiter optimiert werden. Da sich die Bewegung der Sonne und des Mondes relativ zur Erde durch unendliche Reihen darstellen lässt, wurden diese in [Doo21] benutzt, um eine Entwicklung des gezeitenerzeugenden Potentials durchzuführen und so die Bestandteile bzw. Partialtiden zu erhalten - die harmonische Analyse wird dann mit den Kreisfrequenzen der so gewonnenen Partialtiden ausgeführt. Bei einer Zeitreihenanalyse muss man aber immer berücksichtigen, dass die Abtastrate, mit der die Messungen in die Datenreihe aufgenommen worden sind, die Bandbreite an korrekt beschreibbaren Partialtiden beschränkt, wenn man sie nicht vorgibt. Grundlage hierfür ist das Nyquist-Shannon-Abtasttheorem [ET01, S. 382]:

Für die Frequenz einer Partialtide muss $f_{Partialtide} < \frac{1}{2} f_{Sampling}$ gelten, andernfalls kann diese Frequenz aufgrund von Alias-Effekten nicht rekonstruiert werden

In dieser Bachelorarbeit wird mit *Harmonic Analysis* ein Programm betrachtet, welches die harmonische Analyse auf Gezeitenstände, die über einen gewissen Zeitraum an verschiedenen Punkten der Erdoberfläche gemessen wurden, anwendet, um so zu erkennen, welche Faktoren welchen Einfluss auf die Entstehung der Gezeiten haben. Wie bereits im vorherigen Kapitel erwähnt, stammen die Datenpunkte für die Betrachtung in dieser Bachelorarbeit aus der Simulation MPIOM, die Herkunft der Daten ist für *Harmonic Analysis* allerdings unerheblich, solange sie im richtigen Format vorliegen: die Daten sind diskret, liegen entweder als Skalar oder Vektor vor (und ändern ihr Format auch nicht während der Messreihe) und sind zeitlich geordnet [Wikd][ET01]. Für eine genauere Betrachtung des mathematischen Hintergrundes der harmonischen Analyse sei auf den Anhang C (Seite 75) verwiesen.

Anmerkung: Wie im Kapitel *Aufgetretene Probleme* (Seite 42) beschrieben gab es Schwierigkeiten mit der korrekten Installation von Bibliotheken. *Harmonic Analysis* bietet zwei Varianten der harmonischen Analyse an: einmal die Verwendung der Methode kleinster Fehlerquadrate sowie eine vereinfachte Variante der Fourier-Analyse. Da die erste Methode zur Bildung der Matrixinversen auf eine Methode aus der IBM-Bibliothek ESSL[2] zurückgreift, wurde für die Leistungsanalyse im Abschnitt *Leistungsanalyse* (Seite 51) auf die vereinfachte Methode zurückgegriffen, um die Zahl der verwendeten Bibliotheken minimal zu halten.

4.3. Programmablauf

Das Projekt *Harmonic Analysis* besteht insgesamt aus mehreren dutzend Fortran-Dateien, wobei einige Dateien nichts zur eigentlichen Funktionalität der harmonischen Analyse beitragen, sondern z.B. notwendig für die Verwendung von Dateiformaten sind. Für das Verständnis des Programmablaufs sind sie vernachlässigbar und werden daher ausgeblendet. Stattdessen folgt nun ein Überblick über die relevanten Vorgänge.

[2]Engineering and Scientific Subroutine Library, `http://www-03.ibm.com/systems/power/software/essl/` (Zuletzt aufgerufen: 21.11.2014)

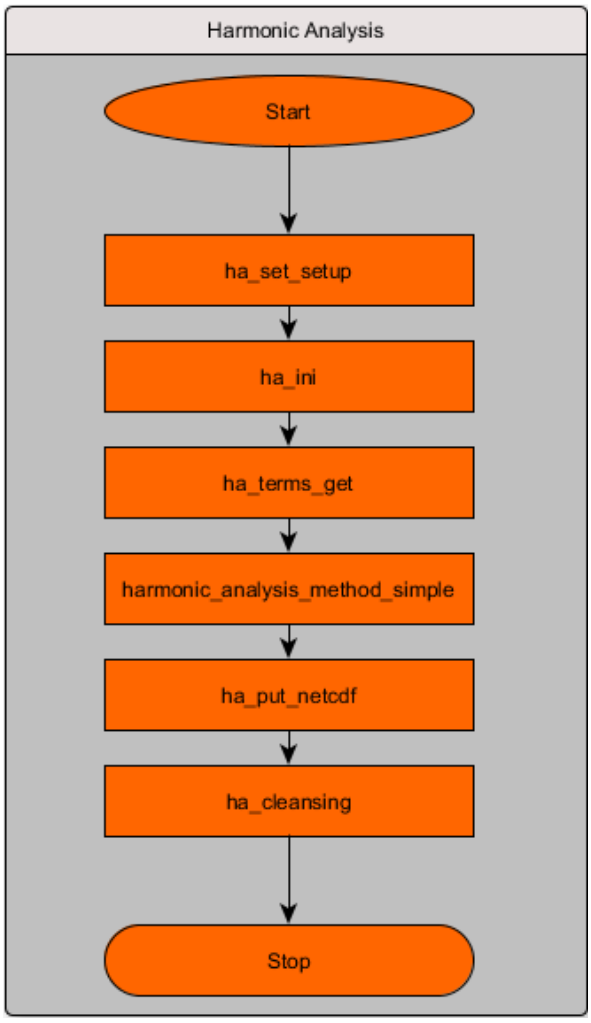


Abbildung 4.2.: Flussdiagramm des Programmablaufs

Dies sind alle wichtigen Methoden, die in der Main-Methode von `harmonic_analysis.f90` ausgeführt werden.

ha_set_setup: Die Konfigurations-Dateien werden eingelesen, um die zu berücksichtigenden Partialtiden und Programmeinstellungen zu laden.

ha_ini: Die Variablen für die Eingabedaten und benötigten Zwischenspeicher werden initialisiert.

ha_terms_get: nähere Beschreibung erfolgt in Abbildung 4.3 (Seite 34)

harmonic_analysis_method_simple: In dieser Methode werden für jeden Ortspunkt über den gesamten Messzeitraum die Amplituden und Phasen mittels der vereinfachten harmonischen Analyse berechnet.

ha__put__netcdf: Die zuvor berechneten Werte für die Amplitude und Phase jeder Partialtide für alle Ortspunkte werden in der Ausgabe-Datei gespeichert.

ha__cleansing: Dynamischer Speicher wird wieder deallokiert und Datei-Streams werden geschlossen

Nach erfolgter Programm-Initialisierung werden also die Simulationsdaten eingelesen, um damit die Partialtiden mittels einer harmonischen Analyse zu berechnen und anschließend zu speichern. Den Abschluss bilden notwendigen Aufräumarbeiten, um das Programm sauber zu einem Abschluss zu führen. Wie sich in Abschnitt *Programmanalyse* (Seite 36) noch zeigen wird, liegt das Bottleneck der Programmlaufzeit in `ha_terms_get`, deshalb wird diese Methode ebenfalls genauer untersucht.

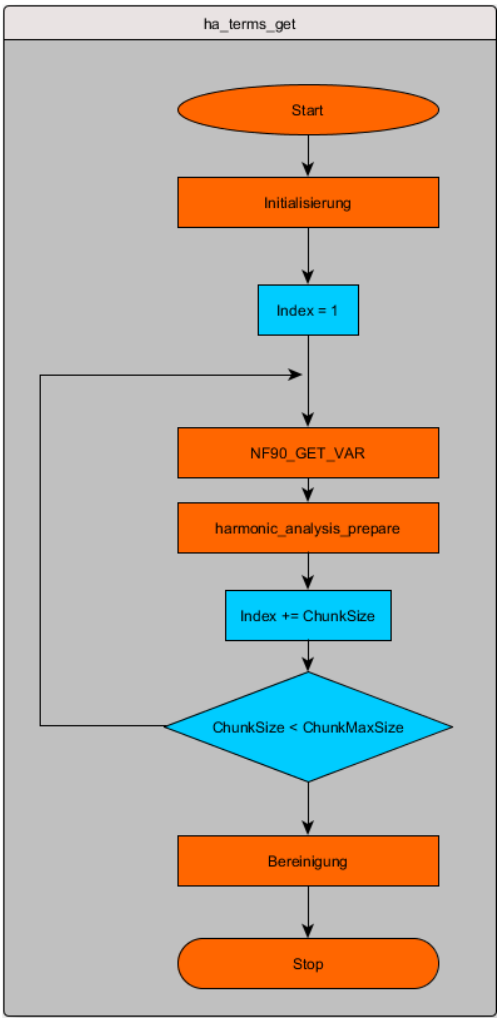


Abbildung 4.3.: Flussdiagramm `ha_terms_get`

Initialisierung: Allokierung und Initialisierung von Speicher für die Koeffizienten der Fourier-Analyse

NF90_GET_VAR: Die Daten eines gesamten Chunks werden aus der netCDF-Eingabe-Datei eingelesen

harmonic_analysis_prepare: Aus den eingelesenen Daten werden die Terme der harmonischen Analyse berechnet, dabei handelt es sich bis auf einen Faktor um die Koeffizienten aus C.3 (Seite 76)

Bereinigung: Deallokierung von nicht mehr benötigten Variablen

Wie in Abbildung 4.3 gezeigt, werden wiederholt die Methoden `NF90_GET_VAR` und `harmonic_analysis_prepare` aufgerufen. `NF90_GET_VAR` lädt aus der Eingabedatei alle Zeitschritte von *index* bis $(ChunkSize - 1)$ in das Array *inData*, welches im Abschnitt *Input* (Seite 29) beschrieben worden ist. Dieses Array wird dann an `harmonic_analysis_prepare` übergeben, welches dann die harmonischen Terme weiterberechnet. Am Ende der Iteration wurde der gesamte Datensatz einmal eingelesen und die zugehörigen harmonischen Terme berechnet. Diese harmonischen Terme werden dann an `harmonic_analysis_method_simple` (vgl. Abbildung 4.2) übergeben, um so die Amplituden und Phasen der betrachteten Partialtiden zu erhalten.

Ein Chunk ist dabei eine Anzahl an Zeitschritten, die auf einmal eingelesen und an die harmonische Analyse übergeben wird, sodass die harmonischen Terme iterativ aufgebaut werden. Ein einzelner Zeitschritt umfasst dabei den Gezeitenstand aller Gitterpunkte. $ChunkMaxSize$ ist die Gesamtzahl an Zeitschritten, die im Datensatz enthalten sind, d.h. für den kleinen Datensatz gilt $ChunkMaxSize = 1488$ und für den großen Datensatz gilt $ChunkMaxSize = 17520$.

4.4. Benötigte Bibliotheken

Harmonic Analysis benutzt neben den Bibliotheken für OpenMP und MPI und diversen anderen Systembibliotheken, zwei weitere Bibliotheken: NetCDF und ESSL. Wie im Abschnitt *Aufgetretene Probleme* (Seite 42) beschrieben, sind Schwierigkeiten aufgetreten, als *Harmonic Analysis* mit der Xeon-Phi-Karte ausgeführt werden sollte. Da es viel Zeit in Anspruch genommen hat, diese Probleme so weit es ging zu beheben oder zu umgehen, wurde auf die Verwendung der Bibliothek ESSl verzichtet. Da die Methode der kleinsten Fehlerquadrate in *Harmonic Analysis* auf eine Funktion dieser Bibliothek zurückgreift, wurde daher stattdessen die Methode `harmonic_analysis_method_simple` verwendet. Diese Methode ist zwar nur eine vereinfachte Form der harmonischen Analyse (vgl. Anhang C, Seite 75), benötigt dafür aber keine Funktionen aus ESSL.

NetCDF hingegen ist unabdingbar für die Funktion von *Harmonic Analysis*, da die Eingabe-Daten im NetCDF-Format gespeichert sind. Letztendlich konnte NetCDF aber doch noch korrekt eingerichtet und somit verwendet werden (vgl. Abschnitt *Aufgetretene Probleme*, Seite 42).

4.5. Programmanalyse

Um *Harmonic Analysis* auf der Xeon-Phi-Karte auszuführen, muss zunächst einmal die Programmstruktur analysiert werden, um geeignete Bereiche im Quellcode zu identifizieren, die auf der Xeon-Phi-Karte ausgeführt werden sollten. Für *Harmonic Analysis* wurde zwar schon in [Rup14, Kapitel 3] eine Analyse der Programmstruktur durchgeführt, die wichtigsten Punkte werden hier aber nochmal dargestellt. Dazu berechnet der Autor die zu erwartende Problemgröße und identifiziert das Bottleneck. Beide Eigenschaften lassen dann Rückschlüsse zu, wie die Xeon-Phi-Karte eingesetzt werden sollte, um die Programmlaufzeit von *Harmonic Analysis* zu verbessern.

Die Problemgröße wird in erster Linie von den Daten bestimmt, die in der Methode `NF90_GET_VAR` aus dem Eingabedatensatz eingelesen und in der Variable *inData* gespeichert werden. *inData* enthält nach dem Einlesen $ChunkSize$ viele Zeitschritte, ein Zeitschritt enthält $362 \cdot 192 \cdot 1$ viele Gezeitenstände, die als Floats abgespeichert werden. Daraus ergibt sich folgende Problemgröße:

$$P(ChunkSize) = ChunkSize \cdot 362 \cdot 192 \cdot 1 \cdot 4Byte = ChunkSize \cdot 271,5\ KB$$

Um die Problemgröße abzuschätzen, betrachten wir nun die Extremstellen:

Minimum: Die Schleife in `ha_terms_get` iteriert über jeden einzelnen Zeitschritt, es gilt also $ChunkSize = 1$. Damit ergibt sich die minimale Problemgröße $P_{min} = 271,5\ KB$.

Maximum: Die Schleife in `ha_terms_get` läuft nur einmal über alle Zeitschritte, es gilt also $ChunkSize = ChunkMaxSize$. Damit ergibt sich mit $ChunkMaxSize_k = 1488$ die maximale Problemgröße $P_{max,k} \approx 394,52\ MB$ für den kleinen Datensatz bzw. mit $ChunkMaxSize_g = 17520$ $P_{max,g} \approx 4,54\ GB$ für den großen Datensatz.

Sowohl für den kleinen als auch den großen Datensatz passt die maximale Problemgröße in den Arbeitsspeicher der Xeon-Phi-Karte bzw. des Hosts (vgl. Abschnitt *Testumgebung*, Seite 50), in dieser Hinsicht muss also keine Rücksicht auf den Wert von $ChunkSize$ genommen werden.

Eine Analyse von *Harmonic Analysis* mit `gprof` enthüllt zudem das Bottleneck des Programms. Hierzu wird *Harmonic Analysis* mit dem Compiler-Flag `-pg` kompiliert und auf dem großen Datensatz mit $ChunkSize = 1000$ ausgeführt, um dann anschließend die Laufzeitmessung zu betrachten:

```
  %   cumulative   self
 time   seconds   seconds          name
 99.17    146.14   146.14   harmonic_analysis_prepare_4d
  0.77    147.27     1.13   __intel_ssse3_rep_memcpy
  0.05    147.34     0.07   __intel_memset
```

```
  0.01    147.35     0.01    harmonic_analysis_cos_sin_alloc_4d
```

Listing 4.2: gekürzte `gprof`-Ausgabe

Listing 4.2 zeigt deutlich, dass die Methode `harmonic_analysis_prepare_4d`, die Bestandteil des Interface `harmonic_analysis_prepare` ist, den größten Anteil an der Laufzeit aufweist - an dieser Stelle könnte die Xeon-Phi-Karte dann eine Verbesserung der Laufzeit bewirken.

4.6. Weitere Arbeiten

Die Bachelorarbeit [Rup14] beschäftigt sich ausführlich mit der nicht-parallelisierten Version von *Harmonic Analysis*. Dabei wird näher auf den Programmhintergrund eingegangen und eine ausführliche Programmanalyse durchgeführt. Im Anschluss werden dann die Konzepte der Parallelisierung mit OpenMP und MPI vorgestellt und auf *Harmonic Analysis* angewendet. Die daraus resultierende, parallelisierte Version von *Harmonic Analysis* wurde auch im Zuge dieser Bachelorarbeit verwendet.

Außerdem befindet sich derzeit eine weitere Bachelorarbeit [Beh14] in Arbeit (Stand: 30.11.2014), die sich näher mit dem mathematischen Hintergrund von *Harmonic Analysis* beschäftigt und Methoden der QR-Zerlegung vorstellt, mit denen das Gleichungssystem aus der Methode der kleinsten Fehlerquadrate gelöst werden kann - damit wird dann auch die Abhängigkeit von ESSL beendet.

Zusammenfassung

Harmonic Analysis verwendet als Eingabedatensatz die Simulationsergebnisse von *MPI-OM*, um mittels einer harmonischen Analyse die Amplituden und Phasen aller Partialtiden zu berechnen, deren Periode an *Harmonic Analysis* übergeben worden sind. Diese Ergebnisse werden dann im NetCDF-Format gespeichert. Die Analyse mit `gprof` hat gezeigt, dass der größte Teil der Gesamtlaufzeit auf die Berechnung der harmonischen Terme abfällt, daher empfiehlt es sich, die Xeon-Phi-Karte an dieser Stelle zum Einsatz zu bringen. Da zudem beide Datensätze vollständig in die Arbeitsspeicher vom Host und der Xeon-Phi-Karte geladen werden können, kann *ChunkSize* beliebig verändert werden.

5. Programmanpassungen

Nachdem die Funktionsweise der Xeon-Phi-Karte und von *Harmonic Analysis* vorgestellt wurden, und im Abschnitt *Programmanalyse* (Seite 36) auch ein möglicher Ansatzpunkt für den Einsatz der Xeon-Phi-Karte identifiziert wurde, wird der Quellcode von *Harmonic Analysis* nun entsprechend angepasst.

5.1. Nativer Modus

Die Anpassungen für den nativen Modus beschränken sich darauf, im Makefile von *Harmonic Analysis* das zusätzliche Compiler-Flag `-mmic` einzusetzen und das Programm neu zu kompilieren. Das gleiche Vorgehen muss auf alle verwendeten Bibliotheken angewendet werden, für die Intel nicht bereits eine vorkompilierte Version zur Verfügung stellt. Im Falle von *Harmonic Analysis* müsste die NetCDF-Bibliothek neu kompiliert werden. Nach erfolgter Kompilierung wären dann die Binärdatei und und die Bibliotheken auf die Xeon-Phi-Karte kopiert und die Binärdatei von *Harmonic Analysis* ausgeführt worden. Wie im nachfolgenden Abschnitt *Aufgetretene Probleme* gezeigt wird, hat dies aber nicht funktioniert.

5.1.1. Aufgetretene Probleme

Wenn man versucht, ein Programm im nativen Modus (nähere Beschreibung in Abschnitt *Nativer Modus*, Seite 16) laufen zu lassen, ohne dass die verwendeten Bibliotheken im Datei-Format vorliegen, welches auf der Xeon-Phi-Karte ausgeführt werden kann, wird die Kompilierung nicht durchgeführt werden. Lässt man sich mittels `file` den Dateityp einer kompilierten Binär-Datei anzeigen, die im nativen Modus auf der Xeon Phi ausgeführt werden kann, ergibt sich als Dateiformat: `ELF 64-bit LSB executable, Intel Xeon Phi coprocessor (k1om)`. Wenn man nun *Harmonic Analysis* mit dem `-mmic`-Flag zu kompilieren versucht, ergibt sich unter anderem folgende Fehlermeldung:

```
[...]
x86_64-k1om-linux-ld: skipping incompatible
   ↪ /home/dkrz/k203089/libs/lib/libnetcdf.so when
   ↪ searching for -lnetcdf
x86_64-k1om-linux-ld: skipping incompatible
   ↪ /home/dkrz/k203089/libs/lib/libnetcdf.a when searching
   ↪ for -lnetcdf
```

```
x86_64-k1om-linux-ld: skipping incompatible
   ↪ /home/dkrz/k203089/libs/lib/libnetcdf.so when
   ↪ searching for -lnetcdf
x86_64-k1om-linux-ld: skipping incompatible
   ↪ /home/dkrz/k203089/libs/lib/libnetcdf.a when searching
   ↪ for -lnetcdf
x86_64-k1om-linux-ld: cannot find -lnetcdf
make: *** [harmonic] Fehler 1
```

Listing 5.1: Nativer Modus, Fehlermeldung

Die Ursache hierfür liegt darin, dass die Xeon Phi ein eigenes Dateiformat erwartet. Hiervon sind allerdings nicht nur die ausführbaren Programmdateien sondern auch die dynamischen Bibliotheken betroffen. Intel liefert mit der Karte auch ein dutzend vorkompilierter Bibliotheken mit (siehe Tabelle 3.1, Seite 17) - jedes andere dynamisch geladene Objekt, welches nicht diesem Lieferumfang beiliegt, für die Programmausführung aber benötigt wird, muss mit Intels Compilern für den nativen Modus kompiliert werden. Für kleine und/oder selbstgeschriebene Bibliotheken ist dies mit Hinzunahme des `-mmic`-Flags beim Kompilieren noch relativ simpel, bei umfangreichen Bibliotheken aus externer Quelle stößt man schnell auf Hindernisse: Wie im Abschnitt *Benötigte Bibliotheken* (Seite 35) aufgeführt, ist *Harmonic Analysis* auf mehrere, externe Bibliotheken angewiesen, die alle für den nativen Modus neu kompiliert werden müssten. Da auf der Xeon-Phi keine Compiler zur Verfügung stehen, müssen die Bibliotheken auf dem Host gebaut werden. Beim Installieren der Bibliothek (u.a. mithilfe eines *configure*-Skripts) werden zunächst Testprogramme ausgeführt, um die Funktion des verwendeten Compilers zu überprüfen. Die mit `-mmic` kompilierten Testprogramme werden dann also auf dem Host ausgeführt, schlagen aber fehl, da die Ziel-Architektur nicht mit der Host-Architektur übereinstimmt.

Zwei mögliche Lösungsansätze wären die folgenden:

- Man könnte die *configure*-Dateien jeder Bibliothek anpassen, um die Kompilierung der Testprogramme an die Host-Architektur anzupassen
- Man könnte für jede Bibliothek einen eigenen Wrapper schreiben, um die Ausführung der Testprogramme durch *configure* abzufangen, der die Testprogramme auf der Xeon-Phi-Karte ausführt und die Ergebnisse zurück auf den Host bringt.

Da *Harmonic Analysis* aber auf sehr umfangreiche, externe Bibliotheken zurückgreift, hätten solche Anpassungsmaßnahmen den Umfang der Bachelorarbeit zu sehr erweitert. Daher verzichtet der Autor auf die Ausführung von *Harmonic Analysis* im nativen Modus und beschränkt seine Untersuchungen auf den Offload-Modus der Karte.

5.2. Offload-Modus

Da Messungen gezeigt haben, dass die meiste Laufzeit für die Methode `harmonic_analysis_prepare` verwendet wird, folgt nun eine kurze Beschreibung der Methode,

von der in Listing 5.2 (Seite 40) ein Auszug zu sehen ist.

Neben der Initialisierung von Variablen geschehen zwei Dinge: Zunächst werden benötigte Cosinus- und Sinus-Terme berechnet. Danach werden zwei mehrfach geschachtelte Schleifen ausgeführt, in der für alle Partialtiden für alle Gitterpunkte über alle Zeitschritte iteriert wird, um so die harmonischen Terme zu berechnen, aus denen später dann die Amplituden und Phasen berechnet werden können.

```
!--- Determine the cosine and sine terms
cos_sin(:,:) = harmonic_analysis_prepare_cossin( &
               constituents%frequency(:)        , &
               samplingrate                     , &
               sizeTimeDim                      , &
               offsetTime                         &
                     )
!--- Determine sum of harmonic terms
!$OMP PARALLEL
DO t = 1, sizeTimeDim
!$OMP DO private(j,i,k) collapse(3) schedule(guided, 4)
  DO d = 1, SIZE( inData, 3 )
    DO j = 1, SIZE( inData, 2 )
      DO i = 1, SIZE( inData, 1 )
        IF ( inData(i,j,d,t) > lbtreshhold ) THEN
          DO k = 1, numConstituents
            ycos_ysin(k,i,j,d)%co = ycos_ysin(k,i,j,d)%co&
                      + inData(i,j,d,t) * cos_sin(k,t)%co
            ycos_ysin(k,i,j,d)%si = ycos_ysin(k,i,j,d)%si&
                      + inData(i,j,d,t) * cos_sin(k,t)%si
          END DO
        END IF
      END DO
    END DO
  END DO
!$OMP END DO
END DO
!$OMP END PARALLEL

!--- Determine sum of input data for harmonic mean
!$OMP PARALLEL
DO t = 1, sizeTimeDim
!$OMP DO private(j, i) collapse(3) schedule(guided, 4)
  DO d = 1, SIZE( inData, 3 )
    DO j = 1, SIZE( inData, 2 )
      DO i = 1, SIZE( inData, 1 )
```

```
37          IF ( inData(i,j,d,t) > lbtreshhold ) THEN
              ymean(i,j,d)%mean = ymean(i,j,d)%mean &
39                                  + inData(i,j,d,t)
            END IF
41        END DO
        END DO
43    END DO
!$OMP END DO
45  END DO
!$OMP END PARALLEL
```

Listing 5.2: `harmonic_analysis_prepare` (Auszug) im Originalzustand

Für ein besseres Verständnis, werden die Schleifenvariablen einmal ihrer Bedeutung zugeordnet:

Variable	Bedeutung
t	Zeitschritt, *sizeTimeDim* entspricht *ChunkSize*
d	Tiefe des Ozeans. Im verwendeten Eingabedatensatz ist die Tiefe überall 1
j	y-Koordinate auf dem Gitter
i	x-Koordinate auf dem gitter
k	k-te Partialtide

Tabelle 5.1.: Zuordnung der Schleifenindizes zu ihrer natürlichen Entsprechung

Um unnötige Kopiervorgänge zu vermeiden, werden beide Schleifen in einen Offload-Block eingebettet. Die Methode `harmonic_analysis_prepare_cossin` wird auf dem Host belassen, da die Zeitmessungen im Abschnitt *Leistungsanalyse* (Seite 51) gezeigt haben, dass sie eine zu kurze Laufzeit aufweist, als dass sich ein Offload lohnen würde.

```
!dir$ offload begin target(mic) INOUT(ycos_ysin,ymean)
   ↪ IN(inData,cos_sin)
2   !--- Determine sum of harmonic terms
    !$OMP PARALLEL
4   DO t = 1, sizeTimeDim
    !$OMP DO private(j,i,k) collapse(3) schedule(guided, 4)
6     DO d = 1, SIZE( inData, 3 )
        [...]
8     END DO
```

```
    !$OMP END DO
10  END DO
    !$OMP END PARALLEL
12
    !--- Determine sum of input data for harmonic mean
14  !$OMP PARALLEL
    DO t = 1, sizeTimeDim
16  !$OMP DO private(j, i,t) collapse(3) schedule(guided, 4)
      DO d = 1, SIZE( inData, 3 )
18      [...]
      END DO
20  !$OMP END DO
    END DO
22  !$OMP END PARALLEL
!dir$ end offload
```

Listing 5.3: `harmonic_analysis_prepare` (Auszug) mit Offload-Direktiven

Die im Offload-Block enthaltenen Schleifen werden dann bei der Programmausführung auf der Xeon-Phi-Karte ausgeführt, während der Host auf die Rückkehr des Aufrufs wartet. Die Variablen *inData* und *cos_sin* wurden vor Ausführung des Offload-Blockes initialisiert und müssen für die Berechnung der harmonischen Terme im Offload-Block mit ihren zugewiesenen Werten auf der Xeon-Phi-Karte zur Verfügung stehen. Nach dieser Rechnung werden ihre gespeicherten Werte aber nicht mehr weiterverwendet, sondern im nächsten Schleifendurchlauf überschrieben. Daher müssen ihre Belegungen, die sich im Laufe der Rechnung auch nicht ändern, nicht zurück auf den Host kopiert werden. Mit `IN(inData,cos_sin)` wird der Xeon-Phi-Karte mitgeteilt, dass diese Variablen beim Zurückkopieren übergangen werden sollen (vgl. Tabelle 3.2, Seite 20).

5.2.1. Aufgetretene Probleme

Nachdem alle notwendigen Anpassungen an *Harmonic Analysis* durchgeführt wurden, um die Berechnung der Terme der harmonischen Analyse auf die Xeon-Phi-Karte auszulagern, trat beim Kompilieren von *Harmonic Analysis* folgender Fehler auf:

```
/home/dkrz/k203089/petra/projects/fortran-utils/modules/
  ↪ netcdf/mod_netcdf_datatypes.f90(45): error #7013: This
  ↪ module file was not generated by any release of this
  ↪ compiler.   [NETCDF]
```

Listing 5.4: Offload-Modus, Compiler-Fehlermeldung

Das Problem hierbei lag darin, dass die NetCDF-Bibliotheken nicht mit `mpiifort` kompiliert worden sind, mit dem nun aber *Harmonic Analysis* kompiliert werden soll.

Die Eingabe von `nc-config -all` ergab dann, dass für die Kompilierung der NetCDF-Bibliothek `gfortran` (GNU Fortran) verwendet worden ist. Zur Lösung dieses Problems hat der Autor dann die neuste Version von NetCDF und die zugehörige HDF5-Bibliothek neu auf dem Host installiert. Eine genauere Beschreibung der Installation und der verwendeten Compiler-Flags ist im Anhang A (Seite 71) einsehbar.

Nachdem die HDF5- und netCDF-Bibliothek auf die beschriebene Weise neu kompiliert worden sind, konnte *Harmonic Analysis* auch kompiliert werden. Die gekennzeichneten Programmbereiche wurden bei Programmausführung auch nachweislich auf der Xeon Phi ausgeführt und der Offload funktionierte auch, wenn in *Harmonic Analysis* entweder OpenMP oder MPI hinzugeschaltet wurde. Sobald allerdings das Programm mit sowohl MPI als auch OpenMP kompiliert werden sollte, trat ein neuer Fehler auf:

```
"ld: warning: libnetcdf.so.6, needed by
   ↪ /home/dkrz/k203089/libs/lib/libnetcdf.so, may conflict
   ↪ with libnetcdf.so.7"
[...]
ld: MPIR_Thread: TLS definition in
   ↪ /sw/centos58-x64/intel/impi/4.1.1.036/intel64/lib/
   ↪ libmpi_mt.so section .tbss mismatches non-TLS
   ↪ definition in
   ↪ /sw/centos58-x64/intel/impi/4.1.1.036/intel64/lib/
   ↪ libmpi.so.4 section .bss
/sw/centos58-x64/intel/impi/4.1.1.036/intel64/
   ↪ lib/libmpi.so.4: could not read symbols: Bad value
make: *** [harmonic] Fehler 1
```

Listing 5.5: Offload-Modus, Hybrid-Fehlermeldung

Das Problem ist, dass der Verlinkungsvorgang beim Kompilieren Standard-Bibliotheken und threadsichere Bibliotheken miteinander zu verbinden versucht, da im ersten Installationsversuch die Bibliotheken nicht threadsicher kompiliert worden sind und Intel für seine MPI-Biliothek eine threadsichere und nicht-threadsichere Variante anbietet. Um das Problem zu beheben, wurden die Compiler-Flags in vielen verschiedenen Versuchen neu gesetzt (vgl. Listing A.3), damit die damit kompilierten Bibliotheken nun auch threadsicher sind. Danach konnte die Programmkompilierung erfolgreich abgeschlossen und *Harmonic Analysis* nun wieder im hybriden MPI-OpenMP-Modus ausgeführt werden.

5.3. Weitere Anpassungen

An *Harmonic Analysis* wurden außer durch die eingefügten Offload-Direktiven noch weitere Veränderungen vorgenommen, die in den nachfolgenden Abschnitten vorgestellt werden.

5.3.1. Vektorisierung

Zum jetzigen Zeitpunkt wissen wir nicht, ob die innersten Schleifen bei der Ausführung auf der Xeon-Phi-Karte vektorisiert wurden. Wie im Abschnitt *Weitere Anpassungsmöglichkeiten* (Seite 23) beschrieben, hat der Autor den Status dieser Schleifen mithilfe des Compiler-Flags `-vec-report=2` untersucht. Zu den innersten Schleifen gibt der Compiler die Meldung „`loop was not vectorized: vectorization possible but seems inefficient`“ aus, er kann also die Korrektheit der Vektorisierung garantieren, nicht aber ihren Nutzen. Um auszuprobieren, ob sich die Laufzeit durch eine erzwungene Vektorisierung der inneren Schleifen verbessert, wird die Vektorisierungs-Direktive eingesetzt:

```
1  !dir$ offload begin target(mic) INOUT(ycos_ysin,ymean)
     ↪ IN(inData,cos_sin)
   !--- Determine sum of harmonic terms
3  !$OMP PARALLEL
   [...]
5    !dir$ simd
     DO k = 1, numConstituents
7             ycos_ysin(k,i,j,d)%co = ycos_ysin(k,i,j,d)%co&
                        + inData(i,j,d,t) * cos_sin(k,t)%co
9             ycos_ysin(k,i,j,d)%si = ycos_ysin(k,i,j,d)%si&
                        + inData(i,j,d,t) * cos_sin(k,t)%si
11    END DO
      [...]
13   !dir$ simd
     DO i = 1, SIZE( inData, 1 )
15        IF ( inData(i,j,d,t) > lbtreshhold ) THEN
            ymean(i,j,d)%mean = ymean(i,j,d)%mean &
17                                + inData(i,j,d,t)
          END IF
19   END DO
     [...]
21 !$OMP END PARALLEL
!dir$ end offload
```

Listing 5.6: `harmonic_analysis_prepare` (Auszug) mit Offload- und Vektorisierungs-Direktiven

Ein Vergleich der Laufzeiten der Programme mit vektorisierten und nicht-vektorisierten Schleifen hat aber ergeben, dass sich die Laufzeit durch diese Maßnahme nicht verbessert hat.

5.3.2. Vorbereitung der Laufzeitanalyse

Alle nachfolgenden Veränderungen am Quellcode haben keine Verbesserung der Laufzeit zur Folge, sondern dienen der Zeitmessung, um am Ende im Kapitel *Durchführung und Auswertung* (Seite 50) den Nutzen der Xeon-Phi-Karte für das Programm *Harmonic Analysis* einschätzen zu können. Um die Auswertung zu erleichtern, wird an neun verschiedenen Stellen im Programm die Zeit gemessen und in Dateien gespeichert. Der Ablauf der Zeitmessung ist in Listing 5.7 dargestellt.

```
#define JANNEK
[...]
#ifdef JANNEK
  CALL GET_ENVIRONMENT_VARIABLE("DATEINAME",datei)
  open(42,file='zeitmessung/42termsGet_'//datei,
    ↪ status='old',action='write',position='append')
#endif
[...]
#ifdef JANNEK
  CALL SYSTEM_CLOCK(clock%temp)
#endif

...Abschnitt, dessen Laufzeit gemessen werden soll...

#ifdef JANNEK
  CALL SYSTEM_CLOCK(clock%cnt1, clock%rate)
  write(42,*) REAL(clock%cnt1 -
    ↪ clock%temp,flt)/REAL(clock%rate,flt)
  close(42)
#endif
```

Listing 5.7: Allgemeiner Ablauf der Zeitmessung

Im Skript, welches das Programm ausführt, wird eine Umgebungsvariable `export DATEINAME='sammlung'$dateiIndex` gesetzt, jeder Lauf mit veränderten Werten für die Threadzahl oder *ChunkSize* bekommt dann einen eigenen Dateiindex zugewiesen, um die spätere Zuordnung zu vereinfachen. Das Symbol `JANNEK` wird verwendet, um die Zeitmessung schnell an- oder auszuschalten. Bei durchzuführender Zeitmessung liest das Programm ab Zeile 4 den Dateinamen ein und öffnet die Datei. In den folgenden Zeilen werden dann die Zeiten gemessen, um diese in Zeile 16 am Ende der Datei hinzuzufügen.

Um darzustellen, wo in *Harmonic Analysis* die Zeitmessungen durchgeführt werden, werden die entsprechenden Bereiche in Abbildung 5.1 markiert und in Tabelle 5.2 nochmal beschrieben. Dabei muss man beachten, dass eine Markierung einer Methode in Abbildung

5.1 nicht automatisch bedeutet, dass auch die gesamte Methode gemessen wird, sondern dass eventuell auch nur ein Teil davon gemessen wird.

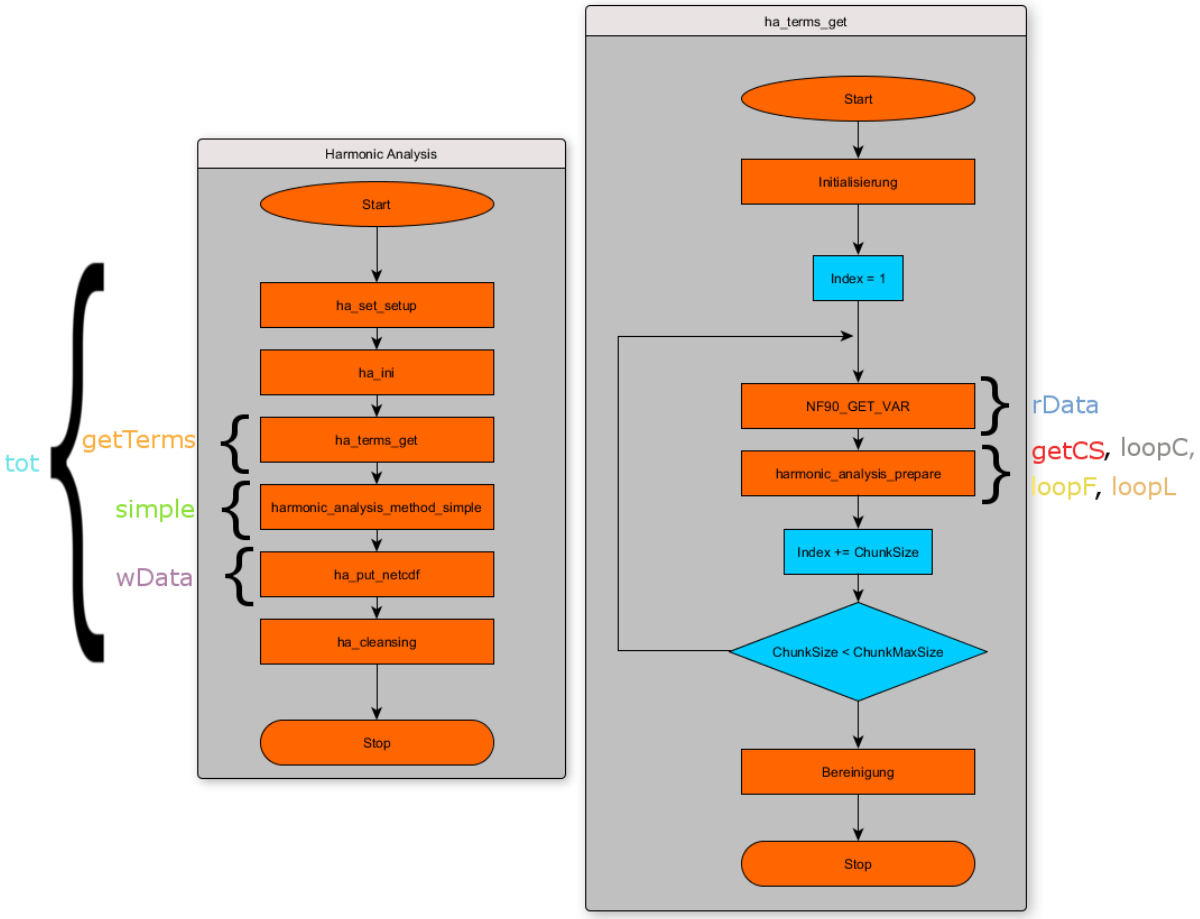


Abbildung 5.1.: Darstellung der Messbereiche

Bezeichnung	Beschreibung
getCS	Berechnung der Cosinus- und Sinusfunktionen der harmonischen Terme
simple	Letzte Rechnungen, um aus den harmonischen Termen die Amplituden und Phasen zu bestimmen
rData	*ChunkSize* viele Zeitschritte werden aus der Eingabedatei eingelesen und in *inData* gespeichert
wData	Speichern der Amplituden und Phasen aller Partialtiden aller Gitterpunkte
getTerms	Laden aller Daten und Berechnung aller harmonischer Terme; beinhaltet rData und loopL
loopC	Berechnung der harmonischen Terme; umschließt den Offload-Bereich; beinhaltet loopF und loopL
loopF	Umfasst die Zeilen 9 bis 28 aus Listing 5.2 (Seite 40)
loopL	Umfasst die Zeilen 31 bis 46 aus Listing 5.2 (Seite 40)
tot	Programmgesamtlaufzeit; beinhaltet alle anderen Zeitmessungen

Tabelle 5.2.: Beschreibung der verschiedenen Messbereiche

Abgesehen von den Anpassungen der Job-Skripte wurden in insgesamt vier verschiedenen Dateien von *Harmonic Analysis* Veränderungen vorgenommen, die der Durchführung des Offload oder der Zeitmessung für die Leistungsanalyse dienen. Im Fall von Veränderungen für das Debuggen wurden diese für eine verbesserte Laufzeit wieder auskommentiert, im Fall der Zeitmessung werden für eine einfachere Zuordnung auch die Bezeichnungen der Messkurven eingefügt.

Datei	Beschreibung	Veränderung
Makefile	test	diverse Debuggingoptionen, z.B. `vec-Report=2`, `-pg`
mod_harmonic_analysis_prepare.f90	beinhaltet die Methoden zur Berechnung harmonischer Terme, vgl. Listing 5.2	Offload-Direktiven, Vektorisierungs-Direktiven, Zeitmessung für *getCS, loopC, loopF, loopL*
mod_ha_terms.f90	Daten einlesen und harmonische Terme berechnen	Zeitmessung für *rData*
harmonic_analysis.f90	enthält die Main-Methode	Zeitmessung für *simple, wData, getTerms, tot*

Tabelle 5.3.: Zuordnung der Schleifenindizes zu ihrer natürlichen Entsprechung

5.4. Korrektheit

Um zu überprüfen, ob die Veränderungen an *Harmonic Analysis* nicht die Korrektheit der Ausgabedaten beeinflusst haben, führte der Autor ein Testskript aus. Dieses Testskript vergleicht den Ausgabedatensatz mit einem Referenzdatensatz, der mit der unveränderten Version von *Harmonic Analysis* erstellt worden ist. Die Abweichung der Amplitude und Phase über alle Gitterpunkte wird am Beispiel der M2-Partialtide berechnet, Abbildung 5.2 zeigt eine Darstellung der Abweichungen aller Gitterpunkte:

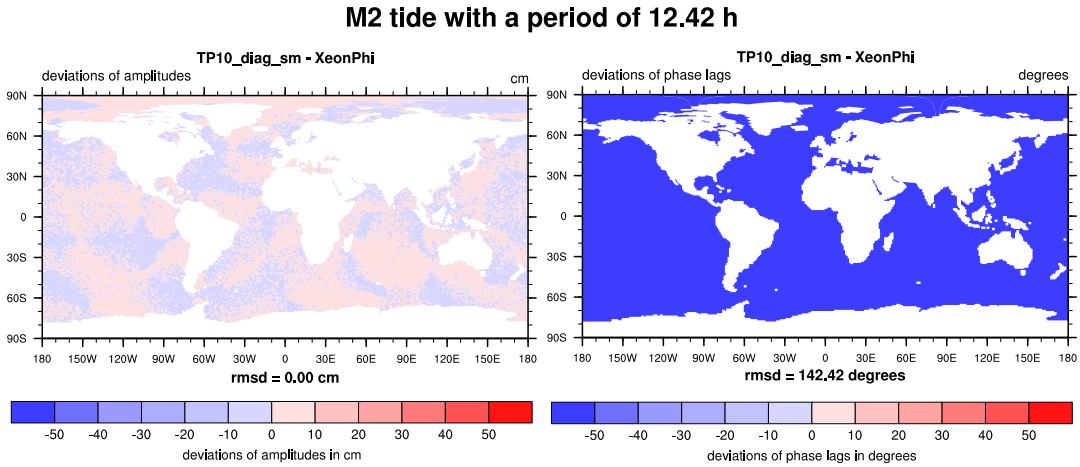


Abbildung 5.2.: Abweichung der Amplituden und Phasen

In der Grafik der Amplitudenabweichung sind minimale Abweichungen zu sehen, die mittlere Abweichung beträgt aber 0 cm. Bei den Abweichungen könnte es sich also um ein rundungsbedingtes Rauschen handeln. Bei der Phase hingegen beträgt die mittlere Abweichung $142,42°$, die zugehörige Grafik zeigt an allen Gitterpunkten eine konstante Abweichung. Dies liegt aber darin begründet, dass das originale Jobskript von *Harmonic Analysis* nach Programmende noch das NCO[1]-Programm `ncatted` aufruft und kleine Veränderungen auf dem Ausgabedatensatz ausführt. Aufgrund der Probleme mit zusätzlichen Bibliotheken, die im Abschnitt *Aufgetretene Probleme* (Seite 42) dargestellt werden, war ein Anliegen, die Zahl verwendeter Bibliothken minimal zu halten. Und da `ncatted` nur eine konstante Änderung der Phase bewirkt, wurde für die Berechnungen, die mit der Xeon-Phi-Karte durchgeführt worden sind, auf die Verwendung von `ncatted` verzichtet. Bis auf eine Konstante stimmt die Phase des Ausgabedatensatzes also mit der

[1]NetCDF Operator, `http://nco.sourceforge.net/` (Zuletzt aufgerufen: 02.12.2014)

Phase des Referenzdatensatzes überein.

Am Beispiel der M2-Partialtide zeigt sich also, dass die Verwendung der Xeon-Phi-Karte für Berechnungen aus *Harmonic Analysis* keinen negativen Einfluss auf die Korrektheit des Ausgabedatensatzes hat.

Zusammenfassung

Die Methode `harmonic_analysis_prepare` wurde mittels Offload-Direktiven so angepasst, dass die Berechnungen der harmonischen Terme automatisch auf die Xeon-Phi-Karte ausgelagert und die Ergebnisse anschließend an den Host zurückgesendet werden - die erzwungene Vektorisierung der Berechnung bringt allerdings keinen Leistungsvorteil. Zudem wurde im Quellcode an verschiedenen Stellen Code eingefügt, um die Laufzeit bestimmter Programmbereiche zu ermitteln, um so den Effekt der Xeon-Phi-Karte auf die Laufzeit detaillierter betrachten zu können. Im Zuge der Anpassungen ist aber auch aufgefallen, dass die Verwendung weiterer Bibliotheken, die nicht von Intel bereitgestellt werden oder sogar schon auf der Xeon-Phi-Karte vorhanden sind, einen nicht unerheblichen Arbeitsaufwand nach sich zieht, da diese neu kompilliert werden müssen.

6. Durchführung und Auswertung

6.1. Testumgebung

Die verwendete Xeon-Phi-Karte wurde bereits im Abschnitt *Technische Daten* (Seite 10) vorgestellt, beim Host handelt es sich um ein *Transtec Phi 4230* [Tra] mit zwei *Intel Xeon E5-2640* Prozessoren (jeweils 6 Kerne à 2,5 GHz mit 2 echten Threads/Kern) und 96GB Arbeitsspeicher. Als Betriebssystem verwendet der Host CentOS6. Die Datensätze werden in Abschnitt *Input* (Seite 29) genauer beschrieben. Der kleine Datensatz ist lokal auf dem Host abgelegt, der große Datensatz hingegen liegt auf dem angebundenen GPFS.

In der nachfolgenden Tabelle werden die Versionsnummern der verwendeten Compiler und Bibliotheken aufgelistet:

Compiler/Bibliothek	Version
Intel MPI Modul	4.1.1.036
Intel Modul	13.5.192
`ifort`	13.1.3
NetCDF	4.3.2
NetCDF-Fortran	4.4.0
HDF5	1.8.19

Tabelle 6.1.: Versionsnummern der verwendeten Compiler/Bibliotheken

Für die Kompilierung von *Harmonic Analysis* wurde `mpiifort` verwendet. `mpiifort` ist ein MPI-Wrapper aus der Intel MPI-Bibliothek, welcher den Intel-Compiler `ifort` verwendet. Vor der Programmausführung bzw. Kompilierung wurde zudem die Laufzeitumgebung angepasst, damit die benötigten Bibliothekspfade eingetragen und die benötigten Compiler geladen werden.

```
. /opt/intel/bin/compilervars.sh intel64
module load intelmpi/4.1.1.036
module load intel/13..5.192
export LD_LIBRARY_PATH=~/libs/lib/:$LD_LIBRARY_PATH
```

Listing 6.1: Konfiguration der Laufzeitumgebung

Der Pfad *~/libs/lib/*, der zu `LD_LIBRARY_PATH` hinzugefügt wird, beinhaltet die Installation der HDF5- und NetCDF-Bibliotheken, die wegen der Probleme, die im

Abschnitt *Aufgetretene Probleme* (Seite 42) beschrieben werden, neu installiert werden mussten.

6.2. Leistungsanalyse

Wie im Abschnitt *Offload-Modus* (Seite 39) beschrieben wird, gibt es zwei unterschiedliche Versionen von *Harmonic Analysis*: eine Version enthält zusätzliche Compiler-Direktiven und wird somit abwechselnd auf dem Host und der Xeon-Phi-Karte ausgeführt; die zweite Version stellt den Originalzustand von *Harmonic Analysis* dar und wird vollständig auf dem Host ausgeführt. Beide Versionen enthalten zusätzlich die benötigten Anweisungen für die Speicherung der verschiedenen Zeitmessungen. In diesem Kapitel werden die Zeitmessungen für beide Programmversionen, unterschiedliche Werte für $ChunkSize$, unterschiedliche Thread-Zahlen und zwei verschieden große Datensätze vorgestellt - die Ergebnisse werden dann miteinander verglichen. Die Beschreibung der Datensätze erfolgt in den Abschnitt *Input* (Seite 29) und *Programmanalyse* (Seite 36).

6.2.1. Durchführung

Harmonic Analysis wird mithilfe der Skripte `startThreadAnalysis.sh` oder `startChunkAnalysis` mit den oben genannten, unterschiedlichen Parametern aufgerufen, die verschiedenen Zeitmessungen werden dann in eigenen Dateien gespeichert. Die Messungen werden für beide Datensätze durchgeführt, indem beide Skripte nach Setzen der Parameter sowohl `job_mpiom_tides_kleinerDatensatz.sh` als auch `job_mpiom_tides_grosserDatensatz.sh`, in denen dann der eigentliche Aufruf von *Harmonic Analysis* durchgeführt wird, aufrufen. Jede Messung einer Konfiguration von Parametern wird dreimal durchgeführt, um Schwankungen abzufedern - die für die verschiedenen Bereiche erhaltenen Laufzeiten werden anschließend mit dem Matlab-Skript `bereinigeDatei.m` gemittelt. Alle Skripte sind auch auf der beigelegten CD enthalten (siehe Anhang D, Seite 79).

Für einen schnellen Überblick, werden in Tabelle 6.2 Kürzel eingeführt, die in den nachfolgenden Abbildungen dann Aufschluss darüber geben, wie viele Threads verwendet wurden, wie groß $ChunkSize$ war und ob die Offload-Direktiven eingebaut waren. Sind die Offload-Direktiven eingebaut, werden Teile von *Harmonic Analysis* auf der Xeon-Phi-Karte ausgeführt; fehlen die Direktiven läuft *Harmonic Analysis* komplett auf dem Host. Da die Zeitmessungen *loopF* und *loopL* (vgl. *Beschreibung der verschiedenen Messbereiche*, Seite 47) innerhalb des Offload-Blockes berechnet wurden, wird in einer Messung auch die Auswirkung dieser beiden Zeitmessungen auf die Laufzeit ausgewertet. In der Tabelle wird daher auch angegeben, ob *loopF* und *loopL* aktiv waren. In den Abbildungen ist in den jeweiligen Titeln auch angegeben, ob Offload-Direktiven (M()) oder *loopF* und *loopL* (S()) aktiv waren.

Wenn $ChunkSize$ variiert, wird die Laufzeit auf $ChunkSize$ aufgetragen und es gilt $ChunkSize \in \{1, 5, 10, 50, 100, 500, 1000, 5000, 10000, 20000\}$.

Wenn die Threadzahl variiert, wird die Laufzeit auf die Anzahl der Threads aufgetragen und es gilt Threads $\in \{1, 2, 3, 5, 15, 30, 60, 120, 180, 240, 300\}$.

Kürzel	*ChunkSize* und Threads	Offload aktiv	*loopF* und *loopL* aktiv
Chunk1	*ChunkSize* variiert, Threads=236	nein	ja
Chunk2	*ChunkSize* variiert, Threads=236	ja	ja
Thread1	*ChunkSize* = 1000, Threads variieren	ja	ja
Thread2	*ChunkSize* = 1000, Threads variieren	ja	nein
Thread3	*ChunkSize* = 1000, Threads variieren	nein	ja

Tabelle 6.2.: Kürzel für gewählte Parameter

6.2.2. Laufzeitvergleich

Die Analyse der Laufzeit wird in diesem Abschnitt nur mit den Ergebnissen der großen Datensätze durchgeführt. Das hat zum einen den Grund, dass der qualitative Verlauf der Laufzeiten mit kleinem Datensatz und der Laufzeiten mit großem Datensatz ähnlich sind, zum anderen wird so in diesem Abschnitt eine bessere Übersicht bewahrt. Die Grafiken der Messungen mit kleinem Datensatz sind aber auf der CD einsehbar.

Zunächst soll festgestellt werden, welchen Einfluss die Zeitmessungen *loopF* und *loopL* im inneren Bereich des Offload-Blockes auf die Laufzeit haben. Hierzu vergleichen wir die Messergebnisse von *Thread1* und *Thread2*.

Threads	300	240	180	120	60
Gesamtlaufzeit *Thread1* [s]	1015,1	548,3	275,4	164,7	136,5
Gesamtlaufzeit *Thread2* [s]	996,2	541,8	265,9	155,6	149,7
Differenz [s]	18,9	6,5	9,5	9,1	-13,2

Threads	30	15	5	3	2	1
Gesamtlaufzeit *Thread1* [s]	138,3	160,7	189,9	279,8	292,8	316,3
Gesamtlaufzeit *Thread2* [s]	149,2	148,9	203,1	309,7	290,3	311,0
Differenz [s]	-10,9	11,8	-13,2	-29,9	2,5	5,3

Tabelle 6.3.: Beeinflussung der Laufzeit durch *loopF* und *loopL*

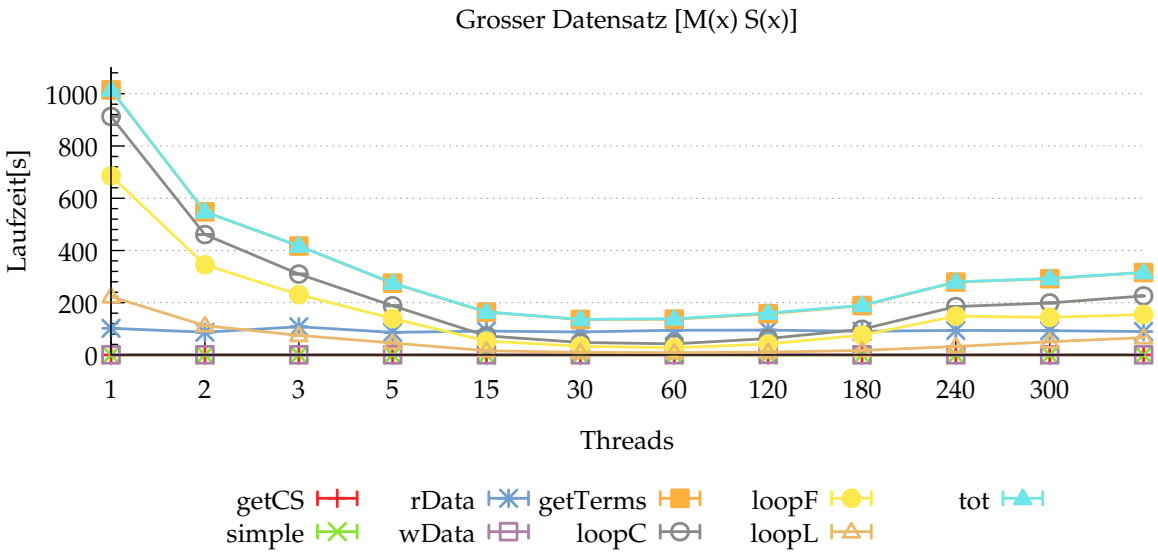


Abbildung 6.1.: *Thread1*, großer Datensatz

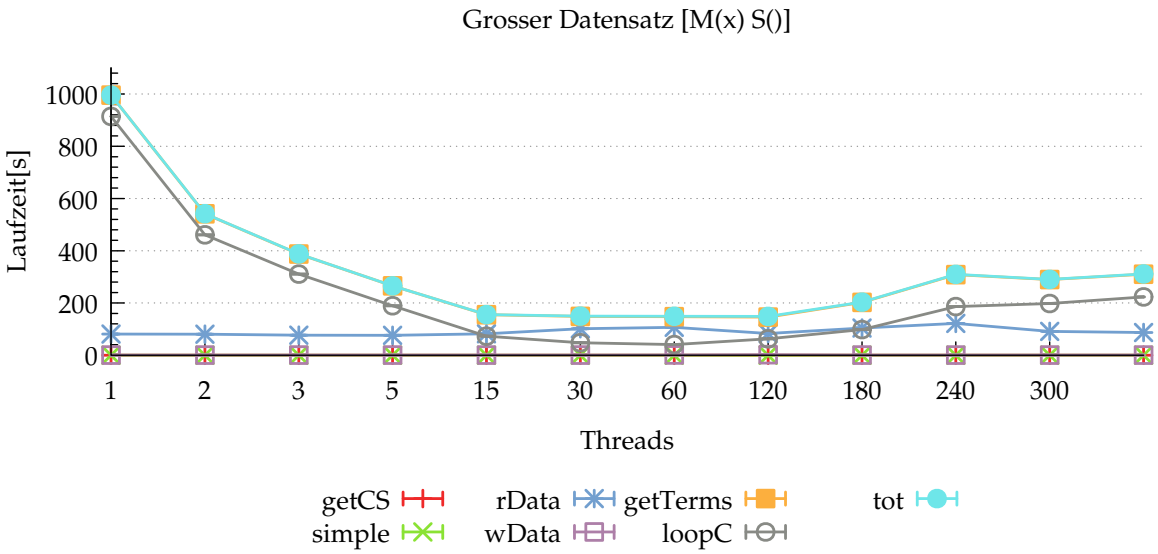


Abbildung 6.2.: *Thread2*, großer Datensatz

Qualitativ betrachtet, ändert die Messung von *loopF* und *loopL* den Verlauf der Zeitmessungen nicht. Auch die gemessene Zeitdifferenz in Tabelle 6.3 ist insgesamt ausgewogen. Das Einschalten der Messungen von *loopF* und *loopL* beeinflusst die Gesamtlaufzeit also nur in vernachlässigbarem Umfang.

Außerdem kann man noch weitere Merkmale aus den Abbildungen 6.1 und 6.2 ablesen:

- Wenn man die absoluten Laufzeiten miteinander vergleicht, erkennt man, dass die Messungen *getCS*, *simple* und *wData* verhältnismäßig kurze Laufzeiten aufweisen. Dies war aber auch so zu erwarten, da der Ausgabedatensatz mit allen 34 Partialtiden auf allen Gitterpunkten nur $8,3\ MB$ groß ist. Verglichen mit der eingelesenen Datengröße wird also nur relativ wenig zurückgeschrieben. Außerdem werden in der Methode `harmonic_analysis_method_simple` nur wenige Rechnungen durchgeführt, der Großteil der harmonischen Analyse wird in `harmonic_analysis_prepare` ausgeführt.
- Die beste Laufzeit mit der Xeon-Phi-Karte ergibt sich im Bereich von 60 bis 30 Threads. Dies ist ein Indiz dafür, dass *Harmonic Analysis* die Xeon-Phi-Karte nicht auslasten kann, da mit weniger Threads die Datengröße, auf der ein Kern Berechnungen durchführt, ansteigt. Und wenn die Laufzeit dann dennoch weiter absinkt, kann der Kern nicht ausgelastet gewesen sein.
- Die Untersuchung mit gprof im Abschnitt *Programmanalyse* (Seite 36) hat gezeigt, dass die meiste Laufzeit auf die Methode `harmonic_analysis_prepare` abfällt. Aus der Grafik können wir nun auch ablesen, dass die geschachtelten Schleifen (vgl. Listing 5.2, Seite 40), die in der Methode enthalten sind, die meiste Laufzeit in Anspruch nehmen - die erste Hälfte von Zeile 1 bis 28 (siehe *loopF*) hat längere Laufzeit als die zweite Hälfte von Zeile 31 bis 45 (siehe *loopL*).

Interessant ist auch noch der Einfluss von $ChunkSize$ auf die Laufzeit. Intuitiv könnte man vermuten, dass $ChunkSize$ bezüglich des verfügbaren Arbeitsspeichers maximal gewählt werden sollte, um so die Anzahl an Kopiervorgängen auf die Xeon-Phi-Karte zu reduzieren. In Abbildung 6.3 erkennt man aber, dass dies nur bedingt stimmt: die Gesamtlaufzeit sinkt deutlich, wenn $ChunkSize$ bis auf 50 hochgesetzt wird, erreicht dort aber mit $t_{ges}(ChunkSize = 50) = 340,6s$ bereits fast die minimale Gesamtlaufzeit von $t_{ges}(ChunkSize = 1000) = 331,7s$ - ab $ChunkSize = 50$ bleibt die Laufzeit nahezu konstant. Daher wurden auch alle weiteren Messungen mit konstantem Wert für $ChunkSize$ und variierender Threadzahl mit $ChunkSize = 1000$ durchgeführt, da sich hiermit die beste Laufzeit ergeben hat.

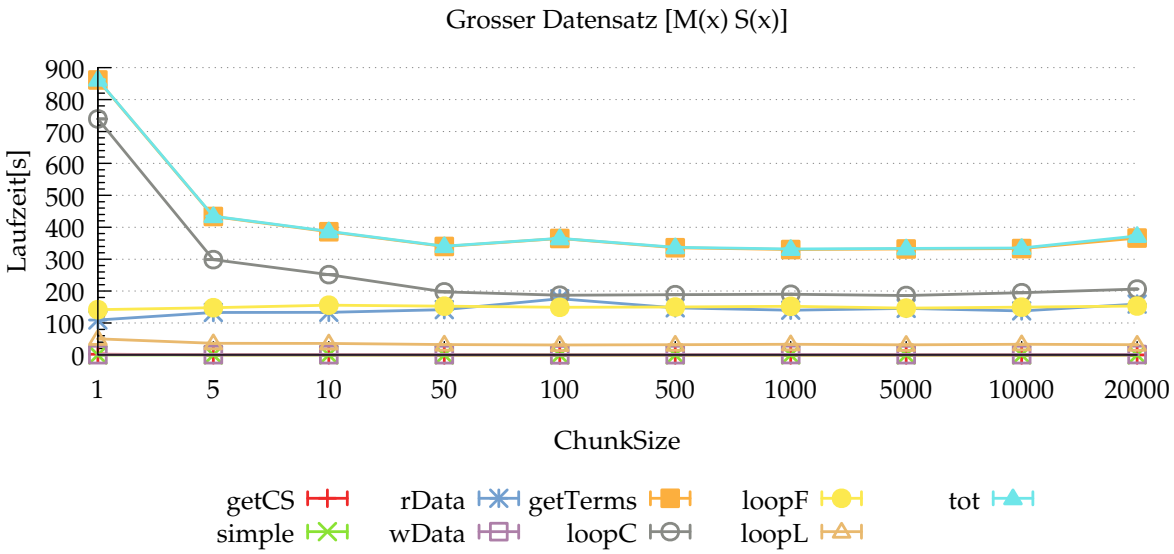


Abbildung 6.3.: *Chunk2*, großer Datensatz

Zum Schluss wird die vermutlich interessanteste Frage geklärt: Läuft *Harmonic Analysis* mit der Xeon-Phi-Karte schneller als ohne?

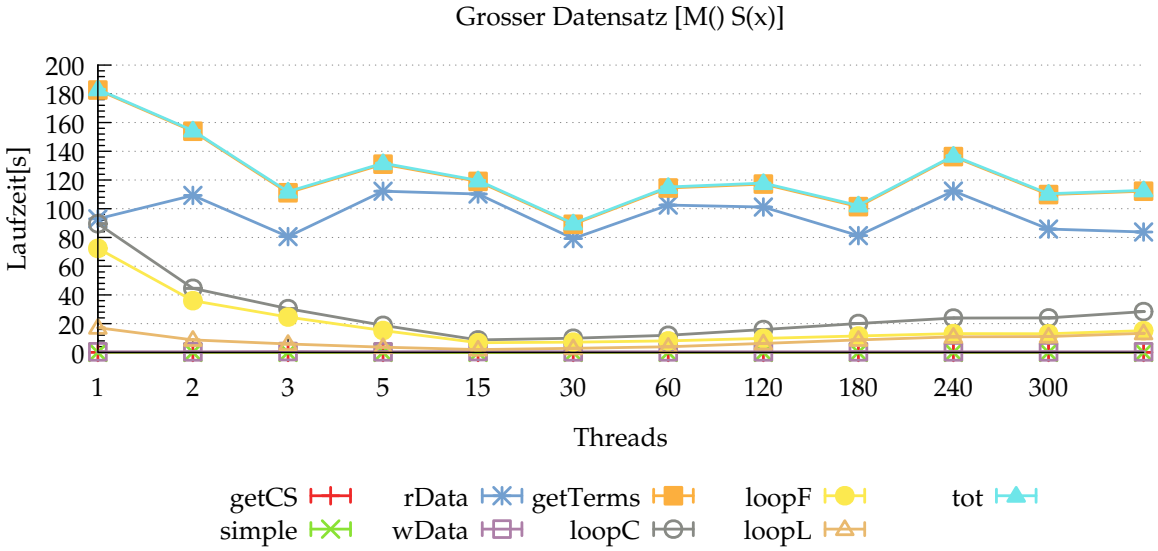


Abbildung 6.4.: *Thread3*, großer Datensatz

Vergleicht man die Laufzeiten aus Abbildung 6.4 mit denen aus Abbildung 6.1 sieht

man sofort, dass *Harmonic Analysis* ohne Offload eine bessere Laufzeit aufweist, als wenn mittels Offload die Berechnungen der harmonischen Terme auf die Xeon-Phi-Karte ausgelagert werden. Die beste Gesamtlaufzeit beträgt ohne Xeon-Phi-Karte ca. $t_{ges}(Threads = 30) = 89,7s$, mit Xeon-Phi-Karte hingegen nur $t'_{ges}(Threads = 30) = 136,5s$. Wenn man berücksichtigt, dass der Host nur 24 echte Threads unterstützt, erreicht *Harmonic Analysis* seine beste Laufzeit vermutlich dann, wenn man eine Threadzahl zwischen 15 und 30 vorgibt. Den Speedup durch die Verwendung der Xeon-Phi-Karte kann man in Abbildung 6.5 ablesen; es wird deutlich, dass *Harmonic Analysis* mit keiner Threadzahl von der Verwendung der Xeon-Phi-Karte profitieren kann.

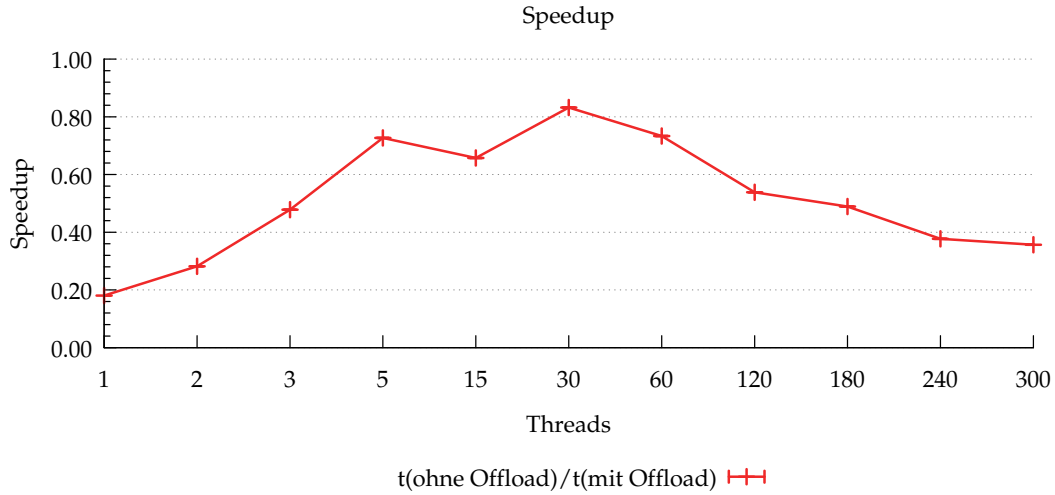


Abbildung 6.5.: Speedup von *Thread3* gegen *Thread1*

Alle Laufzeitmessungen sowie deren Grafiken sind auch auf der CD zu finden. Die Grafiken umfassen dann auch nicht nur die Komplettmessungen sondern auch alle Einzelmessungen.

6.3. Bewertung

Um die Xeon-Phi-Karte besser mit dem Host vergleichen zu können, berechnen wir zunächst seine maximal erreichbare Leistung (vgl. technische Daten im Abschnitt *Testumgebung*, Seite 50):

$$2 \cdot (6\ \texttt{Kerne}\ \cdot 2,5\ \texttt{GHz}\ \cdot 2\ \texttt{Threads/Kern}\ \cdot 8\ \texttt{Operationen}\ ) = 480\ \texttt{GFLOPS}$$

Rein theoretisch steht der Xeon-Phi-Karte, deren maximal mögliche Leistung bei 1008 GFLOPS liegt (vgl. Abschnitt *Zusammenspiel der Bauelemente*, Seite11), dem Host

gegenüber also die doppelte Leistung zur Verfügung. Berücksichtigt man den Speedup, der unter dem Faktor 1 liegt, ist die Verwendung der Xeon-Phi-Karte zur Ausführung von *Harmonic Analysis* hinsichtlich der Effizienz nicht geeignet: Einerseits sind auf dem Host die besseren Laufzeiten erzielt worden und andererseits ist auf ihm eine effizientere Ausnutzung der Ressourcen gegeben als auf der Xeon-Phi-Karte.

Auch mit Blick auf die Anschaffungskosten empfiehlt sich die Ausführung von *Harmonic Analysis* ohne die Verwendung der Xeon-Phi-Karte: Auf Intels Website liegt die Preisempfehlung (für Großabnehmer) für eine Xeon Phi 5110P bei ca. 2500 Dollar [Intg], für einen Intel Xeon E5-2640 Prozessor, von dem zwei im Host verbaut sind, liegt Intels Preisempfehlung bei nur ungefähr 890 Dollar. Auch wenn im Host noch weitere Komponenten wie z.B. Arbeitsspeicher, verbaut werden müssen, liegen die Anschaffungskosten des Hosts somit unter denen für die Xeon-Phi-Karte.

Der Host ist also hinsichtlich seines Anschaffungspreises, der Effizienz und der absoluten Laufzeit der Kombination aus Host und Xeon-Phi-Karte im Offload-Modus überlegen, wenn es um die Ausführung von *Harmonic Analysis* geht.

Auch wenn die Einrichtung des nativen Modus (vgl. *Aufgetretene Probleme*, Seite 38), möglich gewesen wäre, hätte sich vermutlich keine verbesserte Laufzeit ergeben. Der kleine Datensatz hätte dann zwar lokal auf der Xeon-Phi-Karte liegen können, sodass die Kopiervorgänge zur Karte entfallen würden, der große Datensatz passt aber schon nicht mehr in den Arbeitsspeicher der Xeon-Phi-Karte. Spätestens dann wären die Kopiervorgänge also wieder notwendig geworden. Außerdem müsste die Xeon-Phi-Karte dann auch den Quellcode ausführen, der im Offload-Modus immer vom Host ausgeführt wurde, welcher bis auf das parallelisierte Einlesen der Daten aber rein sequentiell aufgebaut ist. Daher hätte sich die Leistungsausbeute der Xeon-Phi-Karte im nativen Modus vermutlich noch weiter verschlechtert.

Die Überlegenheit des Hosts gegenüber der Xeon-Phi-Karte könnte auf zwei Punkte zurückzuführen sein:

1. Die verwendeten Datensätze für die Rechnungen waren zu klein: Wie man in Abbildung 6.3 erkennen konnte, blieb die Laufzeit bei $ChunkSize \geq 50$ ungefähr gleich, obwohl die Zahl der notwendigen Kopiervorgänge mit größerer $ChunkSize$ kleiner wird. Das könnte daran liegen, dass insgesamt zu wenig Daten kopiert werden, sodass ein konstanter Overhead die Laufzeit auf ein Minimum anhebt und die Zeit für die Kopiervorgänge und Berechnungen vernachlässigbar klein sind.

2. Die Vektoreinheiten der Xeon-Phi-Karte wurden nicht ausgelastet: Wie schon im Abschnitt *Vektorisierung* (Seite 44) beschrieben, konnten die Schleifen in `harmonic_analysis_prepare` nicht effizient vektorisiert werden. Das bedeutet, dass das Potential zur Verbesserung der Laufzeit durch Verwendung der großen Vektoreinheit nicht genutzt werden konnte.

Für die die Verwendung der Xeon-Phi-Karte im Offload-Modus spricht hingegen, dass die Berechnungen der Schleifen in `harmonic_analysis_prepare` sehr gut mit hohen Threadzahlen skalieren sollten, da die Amplituden und Phasen eines Gitterpunktes nicht von anderen Gitterpunkten abhängig sind. Somit könnte man ein großes Gitter wählen und die Threads parallel auf verschiedenen Regionen rechnen lassen, ohne dass die Threads miteinander kommunizieren müssten.

6.4. Tatsächliche Zielgruppe

Wie im Abschnitt *Angestrebte Zielgruppe* (Seite 13) beschrieben wurde, sollten die auszuführenden Programme unter anderem sehr gut mit der Threadzahl skalieren und Verwendung für große Vektoreinheiten haben. Programme der Erdsystemforschung können diese Bedingungen zwar durchaus erfüllen, es gilt aber zu überprüfen, ob die Xeon-Phi-Karte weiterhin effizient arbeiten kann, wenn sehr große Datensätze, die in der Erdsystemforschung üblich sind, über die PCIe-Verbindungen verschoben werden und die Programme zusätzlich auch noch verhältnismäßig viel I/O betrieben müssen. Dass die Xeon-Phi-Karte aber durchaus ihre Daseinsberechtigung hat, zeigt sich bereits in der Tatsache, dass sie vom derzeit leistungsstärksten Superrechner der Welt[Top], dem Tianhe-2, im großen Stil verwendet wird: insgesamt sind 48000 Xeon-Phi-Karten vom Typ 31S1P verbaut worden[BR].

Anwender der Xeon-Phi-Karte müssen sich auf eine ausgedehnte Einarbeitungsphase einstellen und berücksichtigen, dass alle Bibliotheken, die ein Programm neben den bereits mitgelieferten (siehe Tabelle 3.1, Seite 17) noch benötigt, neu kompiliert werden müssen - eventuell auch mit eigenständiger Anpassung der Installationsroutine (siehe Abschnitt *Aufgetretene Probleme*, Seite 38). Wenn die Karte aber bereits fertig eingerichtet ist und alle Bibliotheken schon im richtigen Dateiformat vorliegen, wird ein Nutzer auch mit minimaler Einarbeitung schnell Programme auf der Xeon-Phi-Karte ausführen können. Wird nur der native Modus oder der Offload-Modus verwendet, bleibt die Zahl notwendiger Programmanpassungen vermutlich auch sehr übersichtlich, um das Programm zum ersten Mal mit der Xeon-Phi-Karte auszuführen. Das bedeutet dann aber noch nicht, dass das Programm auch bereits optimal läuft. Der Nutzer muss sich dann sehr detailliert mit der Xeon-Phi-Karte auseinandersetzen, um die anfängliche Leistung weiter zu verbessern. Dafür hat man aber mit der Xeon-Phi-Karte unter anderem eine große Vektoreinheit zur Verfügung und muss sich nicht noch zusätzlich mit der Programmierung einer GPU auseinandersetzen[Kin].

Die Xeon-Phi-Karte ist also für Benutzer gedacht, deren Programme die angegebenen Voraussetzungen erfüllen, und die auch bereit sind, sich eingehend mit der Xeon-Phi-Karte zu beschäftigen und die notwendigen Bibliotheken umzubauen.

Zusammenfassung

Die Leistungsanalyse hat gezeigt, dass die Xeon-Phi-Karte hinsichtlich der Effizienz, des Preis-Leistungsverhältnis und der erreichten Laufzeit nicht in Konkurrenz mit den im Host verbauten CPUs treten kann. Zumindest was die Ausführung von *Harmonic Analysis* auf den vorgegebenen Datensätzen anbelangt, ist es von Vorteil, *Harmonic Analysis* komplett auf dem Host auszuführen und nicht auf die Xeon-Phi-Karte teilweise auszulagern. Wenn ein Programm also mit der Xeon-Phi-Karte ausgeführt werden soll, gilt es darauf zu achten, dass es sehr gut mit der Threadzahl skaliert und die Vektoreinheit so gut wie möglich belastet.

7. Abschluss

7.1. Fazit

Der große Vorteil der Xeon-Phi-Karte sind ihre vielen Kerne mit großer Vektoreinheit, die rein theoretisch eine hohe Maximalleistung versprechen. Man muss mit der Xeon-Phi-Karte die auszuführenden Programme also zum Beispiel nicht mehr für die Verwendung auf einer GPU umschreiben, wenn man eine große Vektoreinheit verwenden möchte. Der native Modus und Offload-Modus sind sehr leicht einzurichten, der Offload-Modus bietet zudem noch viele verschiedene Möglichkeiten an, weitere Feineinstellungen am Programm vorzunehmen. Der Nachteil ist nur, dass das in vielen Fällen auch nötig ist: Wenn man das Programm erstmals mit Hilfe der Xeon-Phi-Karte ausführt, muss man gezwungenermaßen an den Feineinstellungen arbeiten, da die Laufzeit wahrscheinlich noch nicht optimal ist. Außerdem könnte die Einrichtung der Karte und benötigter Bibliotheken für Anwender ohne spezielle Kenntnisse ein Problem darstellen. Wenn ein Programm viele Bibliotheken verwendet, die nicht von Intel für die Verwendung mit der Xeon-Phi-Karte vorkompiliert worden sind, müssen diese alle einzeln angepasst werden, was einen sehr hohen administrativen Aufwand nach sich zieht und eventuell auch die Hilfe von externen Personen notwendig macht.

Außerdem ist eine sehr gute Skalierbarkeit mit der Threadzahl eine Eigenschaft, die nicht jedes Programm erfüllen kann; auch eine große Vektoreinheit kann nicht jedes Programm optimal ausnutzen. Erfüllt ein Programm diese Voraussetzungen aber nicht, ist es für eine Ausführung der Xeon-Phi-Karte vermutlich nicht geeignet. Auch das Programm *Harmonic Analysis*, welches in dieser Bachelorarbeit verwendet worden ist, erfüllt nicht alle Voraussetzungen und bleibt bezüglich der Laufzeit somit unter den Möglichkeiten.

7.2. Ausblick

Es gibt noch einige Punkte, die die Xeon-Phi-Karte betreffen und die es wert sind, weiter bearbeitet zu werden:

Vom Aufbau her hat die Xeon-Phi-Karte gewisse Ähnlichkeiten mit Grafikprozessoren ohne aber die Nachteile einer notwendigen Neuprogrammierung mit sich zu bringen. Da GPUs den Ruf haben, sehr energieeffizient zu arbeiten[NVI], stellt sich die Frage, wie energieeffizient die Xeon-Phi-Karte arbeiten kann. Es ist also durchaus möglich, dass sich die Ausführung von *Harmonic Analysis* auf der Xeon-Phi-Karte dann doch lohnt,

wenn man den Fokus auf die erzielte Rechenleistung im Verhältnis zum notwendigen Energieaufwand legt. Dies würde dann aber Strommessungen am System notwendig machen.

Außerdem könnte man den Quellcode von *Harmonic Analysis* nach mathematischen Funktionen durchsuchen, für die es auch eine Implementation in Intels MKL gibt, um sie dann zu ersetzen. Da MKL für das Hochleistungsrechnen optimiert ist und sich auch an die zugrunde liegende Hardware anpasst, könnte so ein Zugewinn an Leistung erzielt werden.

Der native Modus konnte in dieser Bachelorarbeit leider nicht in dem Umfang bearbeitet werden, wie es ursprünglich geplant war. Für eine weitere Beschäftigung mit diesem Betriebsmodus müssten zunächst die Probleme mit der nativen Kompilierung der NetCDF-Bibliothek überwunden werden - erst wenn diese Hürde genommen ist, könnte *Harmonic Analysis* auch im nativen Modus ausgeführt werden. Auf einer einzelnen Xeon-Phi-Karte würde das zwar vermutlich keine verbesserte Laufzeit einbringen, es würde aber etwas erweitern, was ebenfalls ein noch offenes Arbeitsfeld ist: den symmetrischen Modus.

Der symmetrische Modus wurde in dieser Bachelorarbeit nur am Rande erwähnt, da seine Verwendung Veränderungen notwendig macht, die über die Anforderungen des nativen Modus und des Offload-Modus hinausgehen und nicht in den Umfang dieser Arbeit gepasst hätten. Da außerdem nur eine Xeon-Phi-Karte zur Verfügung stand, hätte der symmetrische Modus nicht sein gesamtes Potential ausspielen können. Der symmetrische Modus wird nämlich dann interessant, wenn ein Netzwerk vieler Xeon-Phi-Karten aufgebaut wird. Der symmetrische Modus ermöglicht es dem Anwender dann, Arbeitslasten beliebig über das Netzwerk zu verteilen und so mit MPI das Prinzip des Rechnens auf verteiltem Speicher mittels mehrerer Xeon-Phi-Karten umzusetzen.

Auch die zukünftige Entwicklung der Xeon-Phi-Karte, könnte von großem Interesse sein, denn Intel hat auf der *International Supercomputing Conference* diesen Jahres weitere Details zur neuen Generation der Xeon-Phi-Karte, die bisher unter dem vorläufigen Namen *Knights Landing* bekannt ist, vorgestellt. Nach eigenen Angaben will Intel die Maximalleistung von *Knights Landing* gegenüber der Intel Xeon Phi 5110P auf knapp 3 TFLOPS fast verdreifachen, die Zahl der Vektoreinheiten verdoppeln und weiter die Energieeffizienz steigern. Hinzu kommt, dass *Knights Landing* nicht nur wie die Intel Xeon Phi 5110P als PCI-Express-Karte sondern auch in Form eines gesockelten Prozessors erscheinen soll, welcher dann direkt mit dem Mainboard verbunden werden kann[Hru]. Diese deutlichen Verbesserungen gegenüber der Intel Xeon Phi 5110P würden eine Beschäftigung mit dieser neuen Xeon-Phi-Karte durchaus rechtfertigen.

Literaturverzeichnis

[Amaa] AmandaS. Building a Native Application for Intel Xeon Phi Coprocessors. https://software.intel.com/en-us/articles/building-a-native-application-for-intel-xeon-phi-coprocessors. Zuletzt aufgerufen: 23.11.2014.

[Amab] AmandaS. Building NetCDF* with the Intel compilers. https://software.intel.com/en-us/articles/performance-tools-for-software-developers-building-netcdf-with-the-intel-compilers. Zuletzt aufgerufen: 30.11.2014.

[BBI+] Michaela Barth, Mikko Byckling, Nevena Ilieva, Sami Saarinen, and Michael Schliephake. Best Practice Guide Intel Xeon Phi v1.1. http://www.prace-ri.eu/best-practice-guide-intel-xeon-phi-html/. Zuletzt aufgerufen: 28.11.2014.

[Beh14] Niklas Behrmann. Parallelisierung und Leistungsanalyse numerischer Algorithmen zur Methode der kleinsten Fehlerquadrate. Bachelorarbeit, Universität Hamburg, 2014. In Arbeit (Stand: 30.11.2014).

[Bla] Jordi Blasco. Introduction to Intel Xeon Phi co-processor Workshop. http://www.eresearch.org.nz/sites/www.eresearch.org.nz/files/NeSI_Intel_Phi_workshop-02_new.pdf. Zuletzt aufgerufen: 30.11.2014.

[BR] Anshelm Busse and Jan Richling. Intel's powerful new Xeon Phi coprocessor. http://www.admin-magazine.com/Articles/Exploring-the-Xeon-Phi. Zuletzt aufgerufen: 26.11.2014.

[Bus] BusyBox. http://busybox.net/. Zuletzt aufgerufen: 26.11.2014.

[Chi11] Marco Chiappetta. AMD Breaks 8GHz Overclock with Upcoming FX Processor, Sets World Record. http://hothardware.com/News/AMD-Breaks-Frequency-Record-with-Upcoming-FX-Processor/, 09 2011. Zuletzt aufgerufen: 18.11.2014.

[Cog] Jeff Cogswell. Using Windows Instead of Linux as a Host for Xeon Phi Coprocessor. https://goparallel.sourceforge.net/using-windows-instead-linux-host-xeon-phi-coprocessor/. Zuletzt aufgerufen: 29.11.2014.

[Cou] Rachel Courtland. What Intels Xeon Phi Coprocessor Means for the Future of Supercomputing. http://spectrum.ieee.org/semiconductors/processors/what-intels-xeon-phi-coprocessor-means-for-the-future-of-supercomputing. Zuletzt aufgerufen: 03.12.2014.

[DKKS75] Günter Dietrich, Kurt Kalle, Wolfgang Krauss, and Gerold Siedler. *Allgemeine Meereskunde - e. Einf. in d. Ozeanographie*. Borntraeger, Stuttgart, 1975.

[DKRa] DKRZ. CMIP5 - Das Modell. https://www.dkrz.de/Klimaforschung/konsortial/ipcc-ar5/das-modell. Zuletzt aufgerufen: 24.11.2014.

[DKRb] DKRZ. IBM Power6 "blizzard". https://www.dkrz.de/Klimarechner-en/hpc/ibm-en. Zuletzt aufgerufen: 09.11.2014.

[Doo21] A. T. Doodson. *The harmonic development of the tide-generating potential*. The Royal Society, 1921.

[EK] Elektronik-Kompendium.de. Probleme der modernen Halbleitertechnik. http://www.elektronik-kompendium.de/sites/grd/1011011.htm. Zuletzt aufgerufen: 18.11.2014.

[ET01] William Emery and Richard Thomson. *Data Analysis - Methods in Physical Oceanography*. Elsevier, second and revised edition, 2001.

[FSZ+] Jianbin Fang, Henk Sips, Lilun Zhang, Chuanfu Xu, Yonggang Che, and Ana Lucia Varbanescu. Test-Driving Intel Xeon Phi. http://www.delaat.net/awards/2014-03-26-paper.pdf.

[FVS+] Jianbin Fang, Ana Lucia Varbanescu, Henk Sips, Lilun Zhang, Yonggang Che, and Chuanfu Xu. An empirical Study of Intel Xeon Phi. http://arxiv.org/pdf/1310.5842.pdf. Zuletzt aufgerufen: 08.11.2014.

[Gar] Eric Gardner. Is the Intel Xeon Phi right for me. https://software.intel.com/en-us/articles/is-the-intel-xeon-phi-coprocessor-right-for-me. Zuletzt aufgerufen: 28.11.2014.

[GDD] Everything you need to know about GDDR memory. http://www.maximumpc.com/article/features/everything_you_need_know_about_gddr_memory. Zuletzt aufgerufen: 26.11.2014.

[Hru] Joel Hruska. Intels next-gen Xeon Phi will be 3x faster, include next-gen Hybrid Memory Cube tech. http://www.extremetech.com/extreme/185007-intels-next-gen-xeon-phi-will-be-3x-faster-include-next-gen-hybrid-memory-cube-tech. Zuletzt aufgerufen: 02.12.2014.

[Inta] Hochenetwickelte Leistungseigenschaften für hochparallele Anwendungen. http://www.intel.de/content/www/de/de/processors/xeon/xeon-phi-coprocessor-overview.html. Zuletzt aufgerufen: 28.11.2014.

[Intb] Intel Xeon Phi Coprocessor - Advanced Offload Topics. https://software.intel.com/sites/default/files/Intel%C2%AE_Xeon_Phi%E2%84%A2_Coprocessor_Advanced_Offload_Topics.pdf. Zuletzt aufgerufen: 29.11.2014.

[Intc] Intel Xeon Phi Coprocessor - Architecture Overview. https://software.intel.com/sites/default/files/Intel%C2%AE_Xeon_Phi%E2%84%A2_Coprocessor_Architecture_Overview.pdf. Zuletzt gesehen: 26.11.2014.

[Intd] Intel Xeon Phi Coprocessor - Offloading Computation. https://software.intel.com/sites/default/files/Intel%C2%AE_Xeon_Phi%E2%84%A2_Coprocessor_Offloading_Computation.pdf. Zuletzt aufgerufen: 29.11.2014.

[Inte] Intel Xeon Phi Coprocessor - Optimization and Tuning. https://software.intel.com/sites/default/files/Intel%C2%AE_Xeon_Phi%E2%84%A2_Coprocessor_Optimization_and_Tuning.pdf. zuletzt aufgerufen: 30.11.2014.

[Intf] Intel Xeon Phi Coprocessor - the Architecture. https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner. Zuletzt aufgerufen: 26.11.2014.

[Intg] Intel Xeon Phi Coprocessor 5110P. http://ark.intel.com/de/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core. Zuletzt aufgerufen: 26.11.2014.

[Inth] Intel Xeon Phi Coprocessor: Datasheet. https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-datasheet.html. Zuletzt aufgerufen: 29.11.2014.

[Inti] Intel Xeon Phi Product Family. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html. Zuletzt aufgerufen: 28.11.2014.

[Intj] Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems. https://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems#compiling. Zuletzt aufgerufen: 29.11.2014.

[Kin] Dr. Donald Kinghorn. Top 5 Xeon Phi Misconceptions. http://www.pugetsystems.com/blog/2013/10/03/Top-5-Xeon-Phi-Misconceptions-508/. Zuletzt aufgerufen: 26.11.2014.

[Man11] Markus Mandau. Die Grenzen der PC-Technik. *Chip*, (05), 2011.

[Max] Max-Planck-Institut für Meteorologie. JAMES Special Issue about the MPI-ESM, its components and the CMIP5 simulations. http://www.mpimet.mpg.de/en/science/models/mpi-esm/james-special-issue.html. Zuletzt aufgerufen: 24.11.2014.

[Net] NetCDF FAQ. http://www.unidata.ucar.edu/software/netcdf/docs/faq.html. Zuletzt aufgerufen: 30.11.2014.

[NVI] NVIDIA-Tesla-GPUs beschleunigen weltweit energieeffizientesten Supercomputer. http://www.nvidia.de/object/nvidia-tesla-gpu-power-eurora-supercomputer-june28-2013-de.html. Zuletzt aufgerufen: 02.12.2014.

[PS4] Playstation 4. http://de.playstation.com/ps4/features/techspecs/. Zuletzt aufgerufen: 26.11.2014.

[Rot] Frances Roth. System Administration for the Intel Xeon Phi Coprocessor. https://software.intel.com/sites/default/files/article/373934/system-administration-for-the-intel-xeon-phi-coprocessor.pdf. Zuletzt aufgerufen: 29.11.2014.

[Rup14] Dominik Rupp. Optimization and parallelization of the post-processing of a tidal simulation. Bachelorarbeit, Universität Hamburg, 2014.

[SKC13] Erik Saule, Kamer Kaya, and Ümit V. Catalyürek. Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi. http://arxiv.org/pdf/1302.1078v1.pdf, Februar 2013. Zuletzt aufgerufen: 30.11.2014.

[Top] Tianhe-2. http://www.top500.org/system/177999. Zuletzt aufgerufen: 02.12.2014.

[Tra] Transtec phi 4230. http://shop.transtec.de/doit/loadva/software/w3/CPM_PDC_PDF/PCs-Workstations/CUDA/PH4230CXPD-B_D.pdf. Zuletzt aufgerufen: 29.11.2014.

[Wika] Wikipedia.de. Gezeiten. https://de.wikipedia.org/wiki/Gezeiten. Zuletzt aufgerufen: 08.11.2014.

[Wikb] Wikipedia.de. Gezeitenvorausberechnung. https://de.wikipedia.org/wiki/Gezeitenvorausberechnung. Zuletzt aufgerufen: 09.11.2014.

[Wikc] Wikipedia.de. Postprocessing. https://de.wikipedia.org/wiki/Postprocessing. Zuletzt aufgerufen: 18.11.2014.

[Wikd] Wikipedia.de. Zeitreihenanalyse. https://de.wikipedia.org/wiki/Zeitreihenanalyse. Zuletzt aufgerufen: 20.11.2014.

[Wil] Lucas A. Wilson. Native Computing and Optimization on Intel Xeon Phi. https://portal.tacc.utexas.edu/documents/13601/900205/Cluster_2013_MIC_Native.pdf/671d6403-b974-4027-b3a6-c4a61a433f5c. zuletzt aufgerufen: 30.11.2014.

[WZS+14] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. *High-Performance Computing on the Intel Xeon Phi*. Springer International Publishing, 2014.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhänge

A. Neuinstallation der Bibliotheken

Die HDF5- und NetCDF-Bibliotheken wurden insgesamt zweimal neu installiert, da sich herausstellte, dass der erste Installationsversuch fehlerbehaftet war. Informationen zur Versionsnummer der Bibliotheken sind in Tabelle 6.1 (Seite 50) einsehbar.

Erste Installation

Als ersten Schritt wurden zunächst einige Compiler-Flags gesetzt, die [Amab] entnommen und auf den Host und den verwendeten Compiler angepasst worden sind:

```
export CC=mpiicc
export CXX=mpiicpc
export CFLAGS='-O3 -xHost -ip -no-prec-div
  ↪ -static-intel'
export CXXFLAGS='-O3 -xHost -ip -no-prec-div
  ↪ -static-intel'
export F77=mpiifort
export FC=mpiifort
export F90=mpiifort
export FFLAGS='-O3 -xHost -ip -no-prec-div
  ↪ -static-intel'
export CPP='icc -E'
export CXXCPP='icpc -E'
```

Listing A.1: Compiler-Flags, 1. Versuch

Danach wurde die HDF5-Bibliothek installiert. Da seit NetCDF 4.2 die C- und Fortran-Bibliotheken in getrennten Paketen ausgeliefert werden, wurde anschließend NetCDF 4.3.2 und NetCDF-Fortran 4.4.0 installiert:

```
./configure --prefix=/home/dkrz/k203089/libs
  ↪ --enable-shared --enable-parallel --enable-fortran
make check install
[...]
./configure --prefix=/home/dkrz/k203089/libs
  ↪ --enable-parallel-tests
make check install
```

```
[...]
./configure --prefix=/home/dkrz/k203089/libs
make check install
```

Listing A.2: Bibliotheks-Installation

Zweite Installation

Beim zweiten Versuch ist die Installationsroutine gleich geblieben und entspricht vollständig der Beschreibung in Listing A.2. Die Compiler-Flags wurden nun aber vorher anders gesetzt:

```
export CC=mpiicc
export CXX=mpiicpc
export F77=mpiifort
export FC=mpiifort
export F90=mpiifort
export CPP='icc -E'
export CXXCPP='icpc -E'
export CFLAGS='-openmp
  ↪ -I/home/dkrz/k203089/libs/include'
export CXXFLAGS='-openmp
  ↪ -I/home/dkrz/k203089/libs/include'
export FFLAGS='-openmp
  ↪ -I/home/dkrz/k203089/libs/include'
export LDFLAGS='-openmp -L/home/dkrz/k203089/libs/lib'
```

Listing A.3: Compiler-Flags, 2. Versuch

B. Gezeitentabelle

Die Tabellen B.1 bis B.4 sind ein Auszug aus [DKKS75, S. 402] und zeigen eine Übersicht der wichtigsten Partialtiden, die zur Entstehung der Gezeiten beitragen. Die Partialtiden unterteilen sich dabei in Abhängigkeit ihrer Frequenz in konstante/langperiodische, eintägige, halbtägige und dritteltägige Partialtiden. Spalte 1 enthält die gängige Abkürzung der Partialtide, Spalte 2 gibt den Ursprung der Partialtide aus dem Sonnenpotential (S) oder Mondpotential (M) an, Spalte 3 ist die zugehörige Frequenz und Spalte 4 gibt eine kurze Einordnung zur Entstehung der Partialtide an.

Symbol	Herkunft	Frequenz (°/Std.)	Entstehung
M_0	M	0	Konstante Mondtide
S_0	S	0	Konstante Sonnentide
Sa	S	0,041	Ellipt. Tide 1. Ordnung zu S_0
Ssa	S	0,082	Deklinationstide zu S_0
MSm	M	0,472	Evektionstide zu M_0
Mm	M	0,544	Ellipt. Tide 1. Ordnung zu M_0
MSf	M	1,016	Variationstide zu M_0
Mf	M	1,098	Deklinationstide zu M_0

Tabelle B.1.: Konstante und langperiodische Partialtiden

Symbol	Herkunft	Frequenz (°/Std.)	Entstehung
σ_1	M	12,927	Variationstide zu O_1
Q_1	M	13,399	Ellipt. Tide 1.Ordnung zu O_1
ρ_1	M	13,472	Evektionstide zu 0_1
O_1	M	13,943	Eintägige Haupt-Mondtide
NO_1	M	14,497	Ellipt. Tide 1.Ordnung zu K_1
P_1	S	14,959	Eintägige Haupt-Sonnentide
K_1	M	15,041	Eintägige Haupt-Deklinationstide
K_1	S	15,041	Eintägige Haupt-Deklinationstide
J_1	M	15,585	Ellipt. Tide 1.Ordnung zu K_1
OO_1	M	16,139	Eintägige Deklinationstide 2.Ordnung

Tabelle B.2.: Eintägigige Partialtiden

Symbol	Herkunft	Frequenz (°/Std.)	Entstehung
$2N_2$	M	27,895	Ellipt. tide 2.Ordnung zu M_2
μ_2	M	27,968	Gr. Variationstide zu M_2
N_2	M	28,440	Gr. ellipt. Tide 1.Ordnung zu M_2
ν_2	M	28,513	Gr. Evektionstide zu M_2
M_2	M	28,984	Halbtägige Haupt-Mondtide
L_2	M	29,528	Kl. ellipt. Tide 1.Ordnung zu M_2
T_2	S	29,959	Gr. ellipt. Tide 1.Ordnung zu S_2
S_2	S	30	Halbtägige Haupt-Sonnentide
K_2	M	30,082	Halbtägige Deklinationstide zu M_2
K_2	S	30,082	Halbtägige Deklinationstide zu S_2

Tabelle B.3.: Haltägige Partialtiden

Symbol	Herkunft	Frequenz (°/Std.)	Entstehung
M_3	M	43,476	Dritteltägige Haupt-Mondtide

Tabelle B.4.: Dritteltägige Partialtiden

C. Mathematischer Hintergrund

Dieser Abschnitt zur Beschreibung des mathematischen Hintergrundes der harmonischen Analyse folgt [ET01, S.380-383, S.392-395], welcher für eine noch detailliertere Auseinandersetzung mit diesem Thema empfohlen wird. Um die periodischen Partialtiden der Gezeiten-Daten zu bestimmen, bedient sich *Harmonic Analysis* einer angepassten Form der Fourier-Analyse.

Der Grundgedanke der Fourier-Analyse besteht darin, dass jedes Signal $y(t)$ endlicher Länge, welches unendlich oft wiederholt wird (also periodischen Charakter hat), als Linearkombination von Sinus- und Cosinus-Termen dargestellt werden kann.[1]:

$$y(t) = \overline{y(t)} + \sum_{p=1}^{\infty} [A_p \cos(\omega_p t) + B_p \sin(\omega_p t)] \tag{C.1}$$

$$\begin{aligned} A_p &= \frac{2}{T} \int_0^T y(t) \cos(\omega_p t) dt \\ B_p &= \frac{2}{T} \int_0^T y(t) \sin(\omega_p t) dt \end{aligned}$$

Dabei seien mit $p \in \mathbb{N}^+$ A_p und B_p die Fourier-Koeffizienten und ω_p eine spezifische Kreisfrequenz, die ein ganzzahliges Vielfaches der Fundamental-Kreisfrequenz $\omega_1 = \frac{2\pi}{T}$ ist, wobei T die Gesamtlänge bzw. Periode des Datensignals darstellt. Da der Sinus und Cosinus durch Phasenverschiebung ineinander überführt werden können, kann C.1 auch in kompakter Form geschrieben werden:

$$y(t) = \frac{C_0}{2} + \sum_{p=1}^{\infty} C_p \cos(\omega_p t - \Theta_p) \tag{C.2}$$

$$\begin{aligned} C_p &= \sqrt{(A_p^2 + B_p^2} \\ \Theta_p &= \tan^{-1}\left(\frac{B_p}{A_p}\right) \end{aligned}$$

[1] `http://www.falstad.com/fourier/` (Zuletzt aufgerufen: 20.11.2014) bietet ein Java-Applet, mit dem man für ein besseres Verständnis die Wirkung von Sinus- und Cosinus-Termen auf das resultierende Signal mitverfolgen kann

Da in der Erdsystemforschung oftmals aber keine kontinuierlichen sondern nur endliche viele, diskrete Messwerte vorliegen, vereinfacht sich C.1 dann zu:

$$y(t_n) = \frac{1}{2}A_0 + \sum_{p=1}^{N/2}[A_p \cos(\omega_p t_n) + B_p \sin(\omega_p t_n)] \tag{C.3}$$

$$\begin{aligned} A_p &= \frac{2}{N}\sum_{n=1}^{N} y_n \cos(\omega_p t_n),\ p = 0, 1, 2, \cdots, N/2 \\ B_p &= \frac{2}{N}\sum_{n=1}^{N} y_n \sin(\omega_p t_n),\ p = 1, 2, \cdots, (N/2) - 1 \end{aligned}$$

bzw. zu der kompakten Darstellung:

$$y(t_n) = \frac{C_0}{2} + \sum_{p=1}^{N/2} C_p \cos(\omega_p t_n - \Theta_p)$$

$$C_p = \sqrt{(A_p^2 + B_p^2}, \quad \Theta_p = \tan^{-1}\left(\frac{B_p}{A_p}\right)$$

Dabei wurden N viele Messungen verwendet, die im gleichen Abstand $\Delta t = t_{i+1} - t_i$ mit $1 \leq i \leq N$ aufgenommen worden sind. Die Fourier-Analyse ist also dazu geeignet, für ein Signal, das in gleichmäßigen Abständen abgetastet wurde, die Amplituden der Signalbestandteile mit Frequenz $i \cdot f_1, 1 \geq i \geq N$ zu bestimmen. Der Bestandteil mit maximaler Frequenz, der sich noch korrekt bestimmen lässt, hat dabei die sog. Nyquist-Frequenz aus dem Sampling-Theorem.

Die harmonische Analyse ist ebenfalls eine Fourier-Analyse, verändert aber die Wahl der spezifischen Kreisfrequenzen. Anstatt wie bei der Fourier-Analyse ganzzahlige Vielfache der Fundamentalkreisfrequenz zu benutzen, werden bei der harmonischen Analyse feste Kreisfrequenzen benutzt, die im Allgemeinen nicht äquidistant sind. Dieses Vorgehen bietet sich dann an, wenn die verursachenden Bestandteile bekannt sind, da dann nämlich die Wirkung und Gewichtung der beobachtbaren Verursacher genauer beschrieben werden können. Ansonten würde man nur Informationen über abstrakte Bestandteile erhalten, für die es in der Natur gar keine Entsprechung gibt. Im Fall einer harmonischen Analyse auf Gezeitendaten bietet es sich dann natürlich an, die Kreisfrequenzen der Partialtiden zu benutzen, die zusammengenommen die vermessenen Gezeiten ergeben. Die Gewichtung der einzelnen Frequenzen ergibt sich dann mit C.3 aus der Amplitude C_p, der zugehörige Phasenwinkel Θ_p gibt hingegen die relative Verzögerung an, mit der die Partialtide ihrer natürlichen Ursache folgt. Wenn die Aufzeichnungsdauer dabei ein ganzzahliges Vielfaches aller Frequenzen der untersuchten Partialtiden ist, entspricht die harmonische Analyse auch wieder vollständig der Fourier-Analyse. Da die Fourier-Analyse aber in der Praxis nach endlich vielen Schritten abgebrochen wird, ergibt sich auf jeden

Fall ein Fehler, welcher aber minimiert werden kann. Da bei Gezeiten-Zeitreihen im Allgemeinen die Datenmenge zudem deutlich größer ist als die Menge an Frequenzen, die tatsächlich an der Gezeitenentstehung mitwirken, haben wir es zudem mit einem überbestimmten Gleichungssystem zu tun. Zur Minimierung des Fehlers bzw. Lösung des überbestimmten Gleichungssystems kann z.B. die Methode der kleinsten Fehlerquadrate benutzt werden. Hierzu muss bei N vielen Messungen und M vielen Partialtiden folgendes Gleichungssystem gelöst werden:

$$\vec{z} = D^{-1}\vec{y} \tag{C.4}$$

Die Lösung dieses Gleichungssystem ergibt sich aus der Berechnung folgender Vektoren und Matrix:

$$\vec{y} = \begin{pmatrix} y_{c_0} \\ y_{c_1} \\ y_{c_2} \\ \dots \\ \dots \\ y_{c_M} \\ y_{s_1} \\ \dots \\ y_{s_M} \end{pmatrix}, \vec{z} = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \dots \\ \dots \\ A_M \\ B_1 \\ \dots \\ B_M \end{pmatrix}, \quad \vec{y}, \vec{z} \in \mathbb{R}^{2M+1}$$

$$D = \begin{pmatrix} N & c_1 & c_2 & \dots & c_M & s_1 & s_2 & \dots & s_M \\ c_1 & cc_{11} & cc_{12} & \dots & cc_{1M} & cs_{11} & cs_{12} & \dots & cs_{1M} \\ c_2 & cc_{21} & cc_{22} & \dots & cc_{2M} & cs_{21} & cs_{22} & \dots & cs_{2M} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_M & cc_{M1} & cc_{M2} & \dots & cc_{MM} & cs_{M1} & cs_{M2} & \dots & cs_{MM} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ s_1 & sc_{11} & sc_{12} & \dots & sc_{1M} & ss_{11} & ss_{12} & \dots & ss_{1M} \\ s_2 & sc_{21} & sc_{22} & \dots & sc_{2M} & ss_{21} & ss_{22} & \dots & ss_{2M} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ s_M & sc_{M1} & sc_{M2} & \dots & sc_{MM} & ss_{M1} & ss_{M2} & \dots & ss_{MM} \end{pmatrix}, D \in \mathbb{R}^{(2M+1)\times(2M+1)}$$

Mit $\alpha_i = f_i T$ ergeben sich die Matrixeinträge für D wie folgt:

$$
\begin{aligned}
c_k &= \sum_{n=1}^{N} \cos 2\pi\alpha_k \frac{n}{N} \\
s_k &= \sum_{n=1}^{N} \sin 2\pi\alpha_k \frac{n}{N} \\
cc_{kj} = cc_{jk} &= \sum_{n=1}^{N} [\cos 2\pi\alpha_k \frac{n}{N} \cos 2\pi\alpha_j \frac{n}{N}] \\
ss_{kj} = ss_{jk} &= \sum_{n=1}^{N} [\sin 2\pi\alpha_k \frac{n}{N} \sin 2\pi\alpha_j \frac{n}{N}] \\
cs_{kj} = sc_{jk} &= \sum_{n=1}^{N} [\cos 2\pi\alpha_k \frac{n}{N} \sin 2\pi\alpha_j \frac{n}{N}]
\end{aligned}
$$

Die Vektoreinträge von $\vec{y}$ ergeben sich aus:

$$
\begin{aligned}
y_{c_k} &= \sum_{n=1}^{N} x_n \cos 2\pi\alpha_k \frac{n}{N} \\
y_{s_k} &= \sum_{n=1}^{N} x_n \sin 2\pi\alpha_k \frac{n}{N}
\end{aligned}
$$

Der Ergebnis-Vektor $\vec{z}$ enthält dann die gesuchten Fourierkoeffizienten, mit denen die benötigten Amplituden und Phasen berechnet werden können.

D. CD-Übersicht

Übersicht der Daten, die auf der CD enthalten sind:

BachelorThesis.pdf Das PDF der Bachelorarbeit

Ergebnisse/ Plots der Laufzeitmessungen

HarmonicAnalysis/ Projects- und Scratch-Ordner von *Harmonic Analysis* inklusiver aller Programmanpassungen, die im Laufe der Bachelorarbeit eingebaut wurden, und Jobskripte (ohne Datensätze)

Laufzeitmessungen/ Textdateien aller Laufzeitmessungen, Skripte zur Auswertung der Messergebnisse

M2-Partialtide/ Grafische Darstellung der Amplituden und Phasen der M2-Partialtide

Erklärung

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

Ich bin mit der Einstellung der Bachelor-Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden.

Hamburg, den 03.12.2014 .....