Hochschule Esslingen
University of Applied Sciences

Forschungsprojekt

Tracing-Tool zur Analyse von IO auf HPC-Systemen

im Studiengang Angewandte Informatik der Fakultät Informationstechnik

Wintersemester 2018/2019

Philipp Koester und Johannes Maisch

Datum: 5. März 2019
Betreuer: Prof. Dr.-Ing. Rainer Keller

bw HPC

Ehrenwörtliche Erklärung

Hiermit versicheren wir, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 5. März 2019 _____

Unterschriften

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1

1 Einleitung

Das High-Performance-Computing (HPC) beschäftigt sich mit dem Hochleistungsrechnen. Im Bereich des High-Performance-Computing geht es darum, dass Computer mit möglichst grosser Leistung und m̈oglichst vielen parallelen Prozessen operieren. Dabei ist es unerlässlich, dass zum einen CPU-Berechnungen als auch Speicher-Zugriffe möglichst schnell durchgeführt werden können.

1.1 BWHPC

Die Arbeiten in diesem Projekt werden am Hochleistungscluster BWHPC durchgeführt. Das BWHPC ist ein Hochleistungscluster des Landes Baden Württemberg, welcher an der Universität Karlsruhe steht und für Forschungszwecke eingesetzt wird.

1.2 Mooresches Gesetz

Die Leistung von Prozessoren wird immer schneller. Die Gewschwindigkeit des Wachstums kann durch das sog. Mooresche Gesetz beschrieben werden. Die Definition dieses Gesetzes ist im folgenden gegeben.

> Die Anzahl an Transistoren, die in einen integrierten Schaltkreis festgelegter Grösse passen, verdoppelt sich etwa alle zwei Jahre. [14]

Das Mooresche Gesetz sagt im Umkehrschluss also aus, dass sich die Prozessorleistung etwa alle zwei Jahre verdoppelt. Dieses Wachstum ist in Abbildung 1.1 für Intelprozessoren beispielhaft dargestellt.

Auffallend in Abbildung 1.1 ist, dass die Frequenz der einzelnen Kerne seit einigen Jahren nicht mehr zunimmt. Dies bedeutet, dass das Wachstum nicht mehr durch das Steigern von Leistung, sondern durch das Parallelisieren von CPU-Kernen bestimmt wird. Ein neuer Prozessor hat also nicht mehr Leistung als ein Ã¤lterer, sondern er besteht aus mehr CPU-Kernen. Gerade im Bereich des Hochleistungsrechnen ergibt sich daraus, dass viele Anwendungen parallel ausgeführt werden.

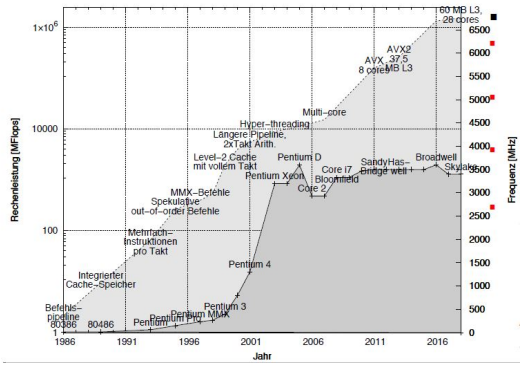

Abb. 1.1: Mooresches Gesetz

1.3 Speicherzugriffe

Mit Speicherzugriffen kann ein Prozessor Daten aus einem Speicher holen und auch in ihn schreiben. Dabei kann im Wesentlichen zwischen Registern, Caches, Hauptspeicher und Festplatte unterschieden werden. Wohingegen der Zugriff auf Register ohne grosse Latenzen möglich ist, ist der Zugriff auf andere Speicher deutlich langsamer. Die Zugriffszeiten sind in Tabelle 1.1 vergleichend dargestellt. Faktor 10 bedeutet hierbei, dass die CPU 10 mal schneller als der Zugriff auf den L3-Cache ist.

Speicher	**Zugriffszeit**
CPU zu L3-Cache	Faktor 10
CPU zu Hauptspeicher	Faktor 100 bis 1000
CPU zu Festplatte	Faktor 1000 bis 1000000

Tab. 1.1: Speicherzugriffe

Aufgrund dieser Geschwindigkeitsunterschiede ist es notwendig den Zugriff auf Speicher möglichst effizient zu gestalten, da es sonst zu erheblichen Engpässen in Programmabläufen kommen kann.

1.4 Dateisysteme

Der Zugriff auf Dateien, welche auf der Festplatte liegen, geschieht über Dateisysteme. Mit einem Dateisystem wird dabei die Ablage dieser Dateien auf der Festplatte organisiert. Damit können diese dann gespeichert, gelesen, verändert oder gelöscht werden. Beim Zugriff auf Dateien kann im Wesentlichen zwischen seriellem und parallelem File-IO unterschieden werden. Die Unterscheidung hierbei liegt darin, wie parallel ausgeführte Programme auf Dateien zugreifen.

1.4.1 Serieller IO

Beim seriellen IO läuft der komplette IO über einen Masterprozess. Dies bedeutet, dass Programme nicht gleichzeitig auf eine Datei zugreifen können. Dies ist in Abbildung 1.2 ersichtlich.

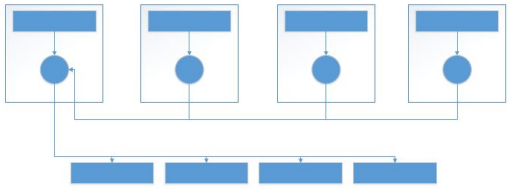

Abb. 1.2: Serial IO [7]

In Abbildung 1.2 ist zu erkennen, dass es zu starken Engpässen kommen kann, wenn mehrere Programme gleichzeitig auf eine Datei zugreifen wollen. Diese Art des IO stellt daher auf kleinen Desktop-Computern mit nur wenigen parallelen Programmen kein Problem dar, im Bereich des High-Performance-Computing mit vielen parallelen Programmen sollte aber auf andere Methoden zurückgegriffen werden.[7]

1.4.2 Paralleler IO

Im Gegensatz zum seriellen IO ist es beim parallelen IO möglich, dass mehrere Prozesse zeitgleich auf eine Datei zugreifen können. Dies ist in 1.3 dargestellt. Darin wird ersichtlich, dass der IO nicht mehr über einen Masterprozess läuft, sondern, dass die einzelnen Prozesse ihren IO unabhängig voneinander durchführen. Der Vorteil hierbei liegt darin, dass die

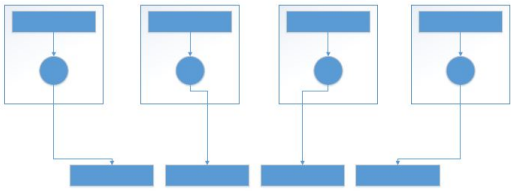

Abb. 1.3: Parallel IO [7]

einzelnen Prozesse parallel auf Dateien zugreifen bzw. in diese schreiben können. Gerade im Bereich des High-Performance-Computing mit sehr vielen parallelen Prozessen stellt dies einen wichtigen Vorteil dar.[7]

1.4.3 POSIX IO

POSIX-IO ist der IO-Part des POSIX-Standards. Der POSIX-Standard ist ein Standard fÜr die Kommunikation von Prozessen mit dem Betriebssystem. POSIX-IO beschreibt dabei verschiedene Funktionen, über welche Programme in einem POSIX-Betriebssystem auf Dateien zugreifen können. Mit diesen Funktionen kann ein Programm dann bspw. eine Datei öffnen, in diese schreiben und diese anschliessend wieder schliessen. Der Zugriff auf POSIX-IO-Funktionen geschieht zumeist über die Glibc. Die Glibc ist eine Bibliothek, welche Systemaufrufe als C-Funktionen bereitstellt. Über diese C-Funktionen können Programme dann Systemaufrufe durchführen.
POSIX-IO eignet sich gut für den Einsatz im Bereich von Desktop-PCs mit nur vergleichsweise wenigen parallelen Prozessen. Für den Einsatz im HPC-Bereich hat POSIX-IO jedoch einige Schwachstellen, welche im Folgenden erläutert werden sollen.

Metadaten

Dateien auf einem POSIX-Dateisystem müssen eine Vielzahl an Metadaten besitzen. Sollen Informationen über eine Datei angezeigt werden, müssen diese Metadaten ausgelesen werden. Auf HPC-Systemen kann dies einen Nachteil darstellen, da das Auslesen der Metadaten von vielen Dateien mitunter sehr lange dauert[11].
Ein weiterer Nachteil der Metadaten liegt darin, dass diese bei jedem Schreibvorgang aktualisiert werden müssen. Dies kostet sehr viel Zeit, wodurch Schreibvorgänge in Bezug auf die Geschwindigkeit erheblich eingeschränkt werden.
Darüber hinaus sind die Metadaten in POSIX-IO sehr unflexibel, da sämtliche Dateien alle Metadaten besitzen müssen und nicht bspw. Metadaten für alle Dateien in einem Ordner gelten können.

File-Descriptoren

Bevor eine Datei in POSIX gelesen werden kann, muss diese zunächst geöffnet werden, um einen File-Descriptor zu erhalten. In diesem File-Descriptor wird der Status der Datei gespeichert. Wird eine Datei wieder geschlossen, wird der File-Descriptor wieder freigegeben.
Der Nachteil von File-Descriptoren kommt zum Tragen, wenn viele Prozesse gleichzeitig auf ein Dateisystem zugreifen wollen. Dann muss das Betriebssystem sehr viele File-Descriptoren parallel verwalten, wodurch bspw. das Öffnen einer Datei immer langsamer wird, je mehr Prozesse parallel auf das Dateisystem zugreifen. In Abbildung 1.4 ist dabei ersichtlich, dass das Öffnen einer Datei linear langsamer wird, je mehr parallele Prozesse auf das Dateisystem zugreifen.

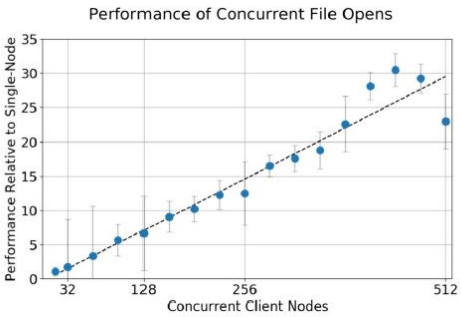

Abb. 1.4: Öffnen einer Datei bei POSIX-IO [12]

Konsistenz

Das Schreiben in eine Datei muss in POSIX konsistent sein. Dies bedeutet, dass das Schreiben die Ausführung einer Applikation so lange blockiert, bis sichergestellt ist, dass ein Lesezugriff das neu geschriebene sieht. Dies hat wiederum den Nachteil, dass im AusfÃ¼hren von Applikationen durch das Schreiben in eine Datei starke Latenzzeiten entstehen. Im HPC-Bereich mit vielen parallelen Applikationen stellt dies ein grosses Problem dar, wenn viele Prozesse gleichzeitig in Dateien schreiben wollen. [12]

1.4.4 MPI-IO

MPIO-IO ist der IO-Part des Message Passing Interface (MPI). MPIO-IO ist dabei eine sog. Middleware, welche i.d.R. nicht direkt von Anwendungen sondern nur indirekt durch höhere Schichten genutzt wird. Es definiert somit einen Standard für parallele IO-Operationen in einer MPI-Applikation. Im Gegensatz zu POSIX-IO ist der Zugriff auf Dateien hierbei nicht Bytestrom- sondern elementorientiert. Der Aufbau einer Datei in MPI-IO ist in Abbildung 1.5 und in Abbildung 1.6 ersichtlich. Eine Datei wird dabei in sog. Fliessen aufgeteilt. Auf diese Fliessen kann über einen Dateityp zugegriffen werden. Ein Dateityp beschreibt ein Muster an Fliessen, welches sich über Teile der Datei oder über die ganze Datei wiederholt. Ein solches Muster ist in Abbildung 1.5 dargestellt. Jede Fliesse im Muster besteht wiederum aus einem elementaren Typ. Der elementare Typ ist der Datentyp über welchen auf die Datei zugegriffen werden kann. Ein Prozess, der auf die Datei über diesen Dateityp zugreift kann somit auf alle Fliessen zugreifen, welche in diesem Muster liegen.

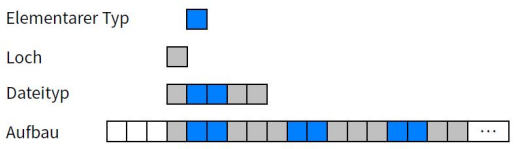

Abb. 1.5: Dateityp bei MPI-IO [9]

In Abbildung 1.6 ist der Aufbau einer Datei aus Sicht von Prozessen dargestellt. Dies wird auch als Dateisicht bezeichnet. Jeder Prozess greift damit über einen anderen Dateityp auf die Datei zu, wodurch mehrere Prozesse gleichzeitig auf die Datei zugreifen können. Dass mehrere Prozesse zeitgleich auf Teile einer Datei zugreifen, ist in dieser Form in POSIX-IO nicht möglich und stellt damit einen entscheidenden Vorteil von MPI-IO gegenüber POSIX-IO dar.

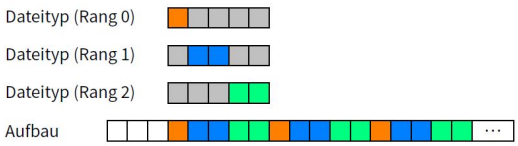

Abb. 1.6: Dateisicht bei MPI-IO [9]

MPI-IO stellt im High-Performance-Bereich eine gute Alternative zu POSIX dar, da damit mehrere Prozesse zeitgleich auf eine Datei zugreifen bzw. in diese schreiben können. Die populärste Implementierung von MPI-IO ist ROMIO. MPI-IO bildet darüber hinaus die Basis vieler IO-Systeme wie bspw. HDF.[8][9]

2 Ziel des Projekts

Ziel des Projekts ist es, eine Software zu entwickeln, die den File-IO von Anwendungen analysiert. Die Software soll sich dabei zwischen das zu analysierende Programm und das Betriebssystem schalten und sämtlichen File-IO abfangen. Der File-IO des Programms soll anschliessend gespeichert und graphisch aufbereitet werden. Die Software soll dabei interaktiv sein. Das bedeutet, es soll mit der Software möglich sein, gezielt nach IO-Engpässen in einer Anwendung zu suchen.

Ziel ist es darüber hinaus, mit der Software ein Framework zur Analyse von File-IO zu schaffen. Dies bedeutet, dass es möglich sein soll, die Software so zu erweitern, dass mit ihr nicht nur POSIX-IO und MPI-IO, sondern z.B. auch Lustre analysiert werden kann.

Im ersten Schritt sollen dabei bestehende Softwarelösungen evaluiert werden. Im zweiten Schritt geht es dann darum, eine eigene Software zu entwickeln, welche die oben genannten Forderungen erfüllt. Die Entwicklung der Software soll dabei portabel mit CMake erfolgen. Die Software soll darüber hinaus Thread-Sicherheit haben. Dies bedeutet, dass sie von mehreren Threads zugleich bedient werden kann. Die Visualisierung soll zudem portabel sein. Die generierten Daten über den File-IO sollen also plattformunabhängig bspw. über einen Webserver visualisiert werden können.

3 Stand der Technik

Im Rahmen der Forschungsarbeit erfolgte zunächst eine Marktrecherche, welche Softwarelösungen zum Tracing von File-IO bereits auf dem Markt sind. Die Lösungen, welche den Anforderungen dieses Projektes am ehesten entsprechen wurden darüber hinaus bezüglicher ihrer Funktionalität evaluiert. Als wichtigstes Kriterium gilt hierbei, dass es mit der Software sowohl möglich ist POSIX-IO zu untersuchen, als auch MPI-IO. Darüber hinaus soll die Analyse zur Laufzeit ohne Recompilieren des Codes möglich sein. Dies soll sowohl für statisch als auch für dynamisch gelinkte Programme der Fall sein.

3.1 Darshan

Darshan ist ein Programm zur Analyse von POSIX-IO und MPI-IO. Mit Darshan kann ein PDF-Report des IO von Programmen erstellt werden. Bei dynamisch gelinkten Programmen ist dies zur Laufzeit möglich, bei statisch gelinkten Programmen ausschliesslich beim Bau des Programms.
Darshan besteht aus zwei Programmen. Mit Darshan-Runtime werden die Informationen über den File-IO eines Programms ermittelt und in einer Log-Datei gespeichert. Die Daten in der Log-Datei können anschliessend mit Darshan-Util dargestellt und analysiert werden.

3.1.1 Funktionsweise

Das Sammeln von Informationen zur Laufzeit von Programmen geschieht über die Systemvariable LD_PRELOAD. Mit dieser ist es möglich Features in ein Prorgamm einzuschleusen. Beim Laden von Shared Libraries wird dabei zunächst nicht die eigentliche Bibliothek geladen, sondern diese, welche unter LD_PRELOAD angegeben wurde. Damit wird dann die Darshan-Bibliothek geladen, welche die IO-Befehle speichert und diese anschliessend an die eigentlichen Bibliotheken weitergibt. Die Funktionsweise von Darshan für dynamisch gelinkte Programme ist in Abbildung 3.1 dargestellt. Die Bibliothek libdarshan.so wird dabei vom zu untersuchenden Programm über LD_PRELOAD geladen. Diese speichert alle MPI-IO- und POSIX-IO-Befehle in Log-Dateien. Diese Log-Dateien können anschliessend mit Darshan-Util ausgewertet werden. Dabei wird entweder ein PDF-Report kreiert in welchem in Diagrammen u.a. dargestellt wird, wieviele File-IO-Operationen jeweils durchgeführt wurden und welche Datenmengen dabei verarbeitet wurden. Alternativ können

die Informationen in eine Textdatei geschrieben und über die Kommandozeile ausgegeben werden.

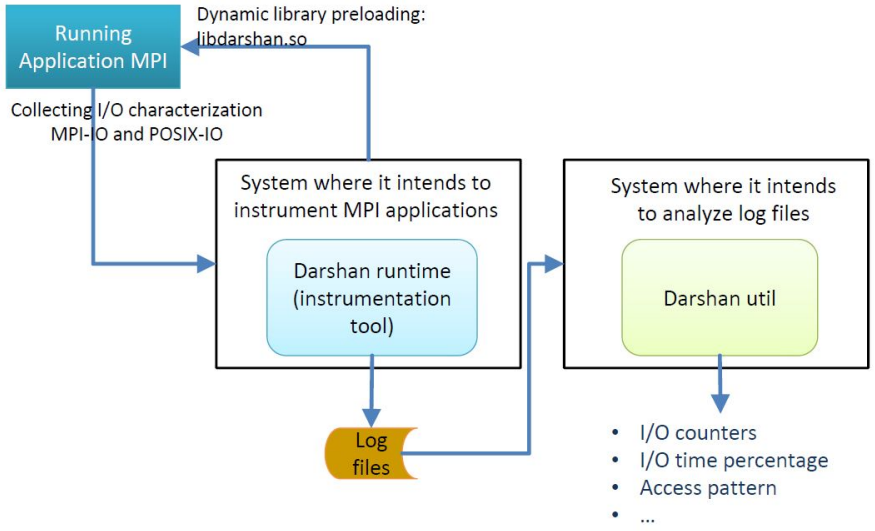

Abb. 3.1: Darshan Aufbau [13]

Das Analysieren von statisch gelinkten Programmen funktioniert ähnlich zu VampirTrace mit Compiler-Wrappern. Diese werden beim Bau anstatt der eigentlichen Compiler aufgerufen. Die Wrapper rufen dabei die eigentlichen Compiler auf, erweitern jedoch das zu kompilierende Programm so, dass es mit Darshan analysiert werden kann. [5][1]

3.1.2 Fazit

Darshan ist ein hervorragendes Programm zur Analyse des IO von dynamisch gelinkten Programmen. Der Nachteil liegt dabei jedoch darin, dass der graphische Output nicht interaktiv ist. Es wird zwar ein PDF-Report kreiert, es ist jedoch nicht möglich interaktiv gezielt nach Schwachstellen im Programm zu suchen. Darüber hinaus können statisch gelinkte Programme mit Darshan nicht zur Laufzeit ohne erneuten Bau untersucht werden, was ebenfalls einen gravierenden Nachteil darstellt.

3.2 VampirTrace

VampirTrace ist ein Programm, welches von der Universität Dresden ursprünglich zur Analyse von MPI-Programmen entwickelt wurde. Mittlerweile ist es ein Tool-Set zur Analyse von parallelen Programmen im HPC-Bereich. Mit VampirTrace können sowohl MPI-IO als auch POSIX-IO untersucht werden. Für die Analyse von Programmen ist es notwendig, diese mithilfe von VampirTrace-Compiler-Wrappern neu zu bauen. Im Makefile müssen

dabei die Compiler durch die Compiler-Wrapper von VampirTrace ersetzt werden. Diese rufen dann wiederum die eigentlichen Compiler auf. Die gebauten Programme können anschliessend zur Laufzeit mit VampirTrace analysiert werden. Eine Untersuchung zur Laufzeit ohne erneuten Bau ist nicht ohne weiteres möglich.

Die gewonnenen Daten werden von VampirTrace in einer Log-Datei im Open-Trace-Format (OTF) gespeichert. Diese Log-Dateien können anschliessend mit Tools, die den Umgang mit OTF beherrschen, visualisiert werden. Am besten eignet sich hierzu das Tool Vampir, welches ebenfalls von der Universität Dresden zu diesem Zweck entwickelt wurde. Mit diesem Tool ist es möglich Daten im OTF-Format interaktiv zu visualisieren und damit gezielt nach Schwachstellen zu suchen. [2][13]

3.3 Score-P

Score-P ist eine Software, die als Nachfolger von VampirTrace entwickelt wurde. In der Funktionsweise ist Score-P VampirTrace dabei recht ähnlich. Die Daten werden ebenfalls im OTF-Format gespeichert und können mit VampirTrace visualisiert werden. Alternativ können die Daten jedoch auch im TAU-Format gespeichert und mit der Software TAU analysiert werden, welche im nächsten Abschnitt erläutert wird. Für die Analyse von File-IO ist VampirTrace aber nach wie vor die bessere Alternative.[10][3]

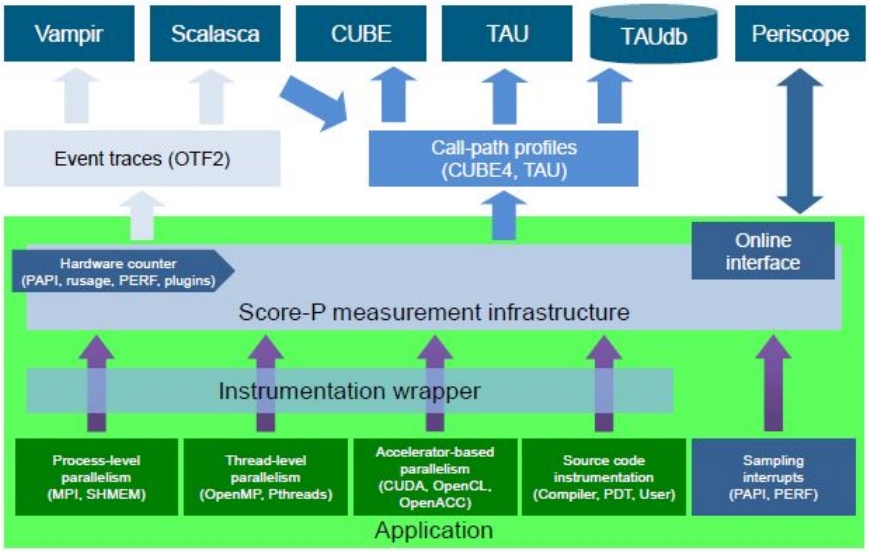

Abb. 3.2: Score-P [3]

3.4 Ludalo

Mit Ludalo ist es möglich, Lustre-Metadaten-Operationen zu analysieren. Dies kann in diesem Projekt im weiteren Verlauf erforderlich sein, wenn die Funktionalität der entwickelten Software für Lustre erweitert werden soll. Lustre ist dabei ein paralleles Dateisystem, welches hauptsächlich im HPC-Bereich eingesetzt wird. [6]

3.5 TAU

Tau ist eine Software, entwickelt von der University of Oregon, zur Analyse von parallelen Applikationen. Eine File-IO-Analyse ist dabei sowohl für POSIX-IO, als auch für MPI-IO möglich. Die von TAU generierten Daten können im OTF-Format gespeichert und anschliessend mit Vampir visualisiert werden. Damit Ä¤hnelt TAU stark VampirTrace, wo die Daten ebenfalls mit Vampir visualisiert werden können. Die Analyse der Programme erfolgt entweder durch das Recompilieren des Quelltextes oder durch das Laden einer Bibliothek mit LD_PRELOAD, was jedoch nur bei dynamisch gelinkten Programmen möglich ist. Dies stellt auch den entscheidenden Vorteil von TAU gegenüber VampirTrace dar. Die Visualisierung ist bei beiden Tools identisch, allerdings können mit TAU dynamisch gelinkte Programme ohne Rekompilieren analysiert werden. [15][16][4]

3.6 Fazit

Keines der untersuchten Programme enthält alle Features, welche in diesem Projekt gewünscht sind. Diese sind in Tabelle 3.1 vergleichend dargestellt. Die Analyse von POSIX-IO

	Darshan	**VampirTrace**	**TAU**
Analyse von POSIX-IO	+	+	+
Analyse von MPI-IO	+	+	+
Interaktive Bedienung	-	+	+
dynamisch gelinkte Programme	+	-	+
statisch gelinkte Programme	-	-	-
Schwerpunkt auf File-IO-Analyse	+	-	-

Tab. 3.1: Marktrecherche

und MPI-IO ist mit allen untersuchten Produkten möglich. Hinsichtlich der Visualisierung sind sich VampirTrace und TAU ähnlich. Bei beiden werden die Daten im OTF-Format gespeichert und können mit Vampir untersucht werden, womit auch eine interaktive Bedienung möglich ist. Mit Darshan können zwar ebenfalls POSIX-IO und MPI-IO analysiert werden, allerdings kann die Visualisierung durch den PDF-Report oder eine Textdatei

nicht interaktiv bedient werden. Der entscheidende Vorteil von Darshan gegenüber den anderen Tools ist jedoch, dass es ausschliesslich für die Analyse von File-IO entwickelt wurde und dadurch deutlich leichtgewichtiger ist.
Die Schwachstelle der Produkte liegt in der Analyse von statisch gelinkten Programmen. Dies ist zwar prinzipiell bei allen möglich, jedoch nur durch das erneute Kompilieren des Quelltextes. Aus diesem Grund soll in diesem Projekt eine eigene Software entwickelt werden, bei welcher dies möglich ist und die eine interaktive Visualisierung beinhaltet.

4 Entwicklung einer eigenen Software

4.1 Architektur

4.2 Umsetzung

5 Aktueller Stand

6 Zusammenfassung und Ausblick

A Anhang

Literaturverzeichnis

[1] Darshan-util installation and usage, 19.01.2019.

[2] VampirTrace 5.14.4: User Manual, 2016.

[3] Score-P: User Manual, 2018.

[4] TAU User Guide, 2018.

[5] Darshan-runtime installation and usage, 22.01.2019.

[6] Holger Berger. Ludalo, 2014.

[7] John Cazes and Ritu Arora. Introduction to Parallel I/O, 26.09.2013.

[8] Peter et.al. Corbett. MPI-IO: A Parallel File I/O Interface for MPI, 1995.

[9] Michael Kuhn. MPI-IO: Hochleistungs-Ein-/Ausgabe, 2016.

[10] Julian Kunkel, Philipp Carns, and Michael Kluge. BoF: Analyzing Parallel I/O, 2014.

[11] Jeffrey B. Layton. POSIX IO Must Die!, 2010.

[12] Glenn Lockwood. What's So Bad About POSIX I/O?, 2017.

[13] Sandra Mendez. Analyzing the High Performance Parallel I/O on LRZ HPC systems, 23.06.2016.

[14] Robert Schanze. Mooresches Gesetz: Defintion und Ende von Moore's Law – Einfach erklärt, 25.02.2016.

[15] Sameer Shende. TAU PERFORMANCE ANALYSIS, 2017.

[16] Sameer Shende, Allan D. Mallony, Wyatt Spear, and Karen Schuchardt. Characterizing I/O Performance Using the TAU Performance System, 2011.