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: 18. Februar 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 18. Februar 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 sind.

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. [11]

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. Gerade im Bereich des Hochleistungsrechnen ergibt sich daraus, dass viele Anwendungen parallel ausgeführt werden.

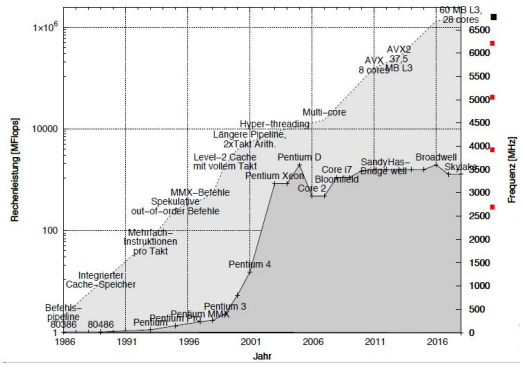

Abb. 1.1: Mooresches Gesetz

1.3 Speicherzugriffe

Mit Speicherzugriffen kann ein Prozessor Daten aus einem Speicher holen und auch in ihn schreiben. Dabei kann grob 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.

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

Tab. 1.1: Speicherzugriffe

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

1.4 Dateisysteme

Der Zugriff auf Speicher geschieht über Dateisysteme. Ein Dateisystem ist notwendig, damit Programme in einem Betriebssystem auf Dateien zugreifen können, welche auf der Festplatte liegen. Bei Dateisystemen kann im Wesentlichen zwischen seriellen und parallelen Dateisystemen unterschieden werden. Die Unterscheidung hierbei liegt darin, wie parallel ausgeführte Programme auf Dateien zugreifen.

1.4.1 Serielles 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.

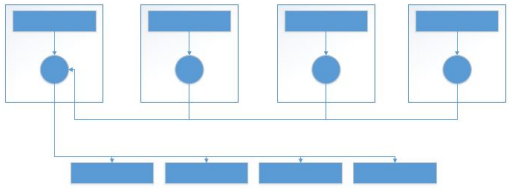

Abb. 1.2: Serial IO [5]

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.[5]

1.4.2 Paralleles 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

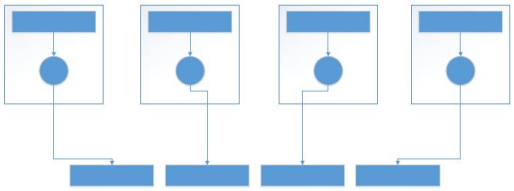

Abb. 1.3: Parallel IO [5]

einzelnen Prozesse parallel auf Dateien zugreifen können. Gerade im Bereich des High-Performance-Computing mit sehr vielen parallelen Prozessen stellt dies einen wichtigen Vorteil dar.[5]

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 Speicher zugreifen können. POSIX-IO ist eine Form des seriellen IO, was im Bereich des High-Performance-Computings zu Problemen führen kann, welche im Folgenden kurz 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[8].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. Mit diesem File-Descriptor wird sichergestellt, dass immer nur ein Prozess auf eine Datei zugreifen kann. 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.

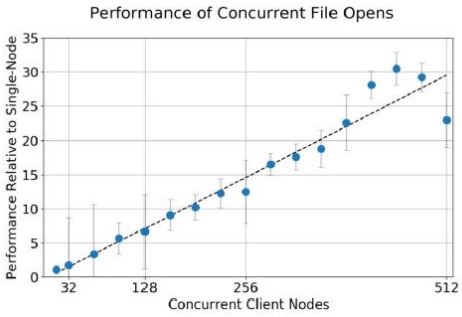

Abb. 1.4: Öffnen einer Datei bei POSIX-IO [9]

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. [9]

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.

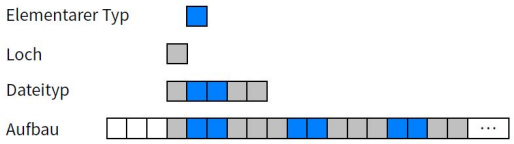

Abb. 1.5: Dateityp bei MPI-IO [7]

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.

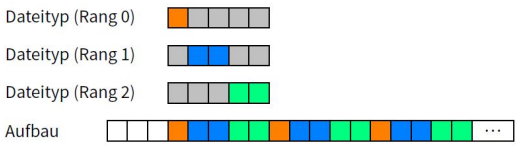

Abb. 1.6: Dateisicht bei MPI-IO [7]

MPI-IO stellt im High-Performance-Bereich eine gute Alternative zu POSIX dar, da damit mehrere Prozesse zeitgleich auf eine Datei zugreifen können. Die populärste Implementierung von MPI-IO ist ROMIO. MPI-IO bildet darüber hinaus die Basis vieler IO-Systeme wie bspw. HDF.[6][7]

2 Ziel des Projekts

Ziel des Projekts ist es, eine Software zu entwickeln, die den IO von Anwendungen analysiert. Die Software soll sich dabei zwischen das zu analysierende Programm und das Betriebssystem schalten und sämtlichen IO abfangen. Der 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.

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. Anforderungen an die Software sind sowohl eine portable Entwicklung, als auch Thread-Sicherheit.

3 Stand der Technik

Im Rahmen der Forschungsarbeit erfolgte zunächst eine Marktrecherche, welche Softwarelösungen zum Tracing von IO bereits auf dem Markt sind. Darüber hinaus wurden diese Lösungen bezüglich ihrer Funktionalität evaluiert. Wichtigstes Kriterium bei der Marktrecherche ist, dass es sowohl möglich ist POSIX-IO zu untersuchen, als auch MPI-IO. Darüber hinaus soll die Analyse zur Laufzeit ohne Recompilieren des Codes möglich 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 IO eines Programms gesammelt und gespeichert, mit Darshan-Util werden diese aufbereitet und dargestellt.

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 ein PDF-Report kreiert in welchem in Diagrammen u.a. dargestellt wird, wieviele IO-Operationen jeweils durchgeführt wurden und welche Datenmengen dabei verarbeitet wurden.

Das Analysieren von statisch gelinkten Programmen funktioniert ähnlich wie bei Vampir-Trace mit Compiler-Wrappern. Diese werden beim Bau anstatt der eigentlichen Compiler

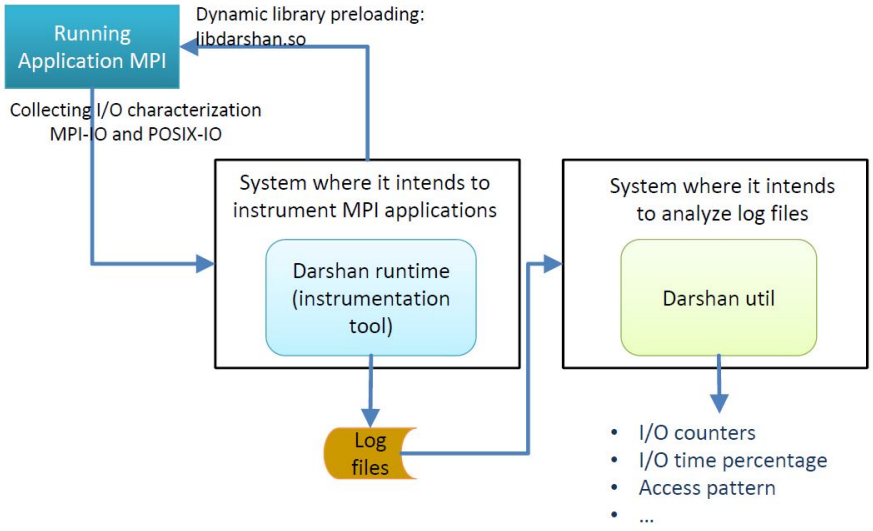

Abb. 3.1: Darshan Aufbau [10]

aufgerufen. Die Wrapper werten das Programm dann aus und schreiben die Log-Dateien. Anschliessend werden die eigentlichen Compiler aufgerufen. [3][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 zur ursprünglich Analyse von MPI-Programmen entwickelt wurde. Mittlerweile ist es ein Tool-Set zur Analyse von parallelen Programmen im HPC-Bereich. Mit VampirTrace kann 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 neuen 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. [2]

3.3 Ludalo

Mit Ludalo ist es möglich Lustre-Metadaten-Operationen zu analysieren. [4]

3.4 TAU

Tau ist eine Software, entwickelt von der University of Oregeon, zur Analyse von parallelen Applikationen. Eine 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. Die Analyse erfolgt entweder durch das Recompilieren des Codes oder durch das Laden einer Bibliothek mit LD_PRELOAD. Hinsichtlich der Funktionalität wurde TAU in diesem Projekt nicht weiter evaluiert. [12][13]

3.5 Fazit

Keines der untersuchten Programme enthält alle Features, welche in diesem Projekt gewünscht sind. Darshan trifft die Anforderungen jedoch am ehesten. Damit kann sowohl MPI-IO als auch POSIX-IO analysiert werden. Allerdings ist dabei keine interaktive Bedienung möglich. Dasselbe ist bei VampirTrace auch der Fall. Damit kann zwar ebenfalls MPI-IO und POSIX-IO analysiert werden, jedoch ist auch hierbei keine interaktive Bedienung vorhanden. Aus diesem Grund soll in diesem Projekt eine Software entwickelt werden, welche die Features von Darshan und VampirTrace mit einer interaktiven Bedienung vereint.

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] Darshan-runtime installation and usage, 22.01.2019.

[4] Holger Berger. Ludalo, 2014.

[5] John Cazes and Ritu Arora. Introduction to parallel i/o, 26.09.2013.

[6] Peter et.al. Corbett. Mpi-io: A parallel file i/o interface for mpi, 1995.

[7] Michael Kuhn. Mpi-io: Hochleistungs-ein-/ausgabe, 2016.

[8] Jeffrey B. Layton. Posix io must die!, 2010.

[9] Glenn Lockwood. What's so bad about posix i/o?, 2017.

[10] Sandra Mendez. Analyzing the high performance parallel i/o on lrz hpc systems, 23.06.2016.

[11] Robert Schanze. Mooresches gesetz: Defintion und ende von moore's law – einfach erklärt, 25.02.2016.

[12] Sameer Shende. Tau performance analysis, 2017.

[13] Sameer Shende, Allan D. Mallony, Wyatt Spear, and Karen Schuchardt. Characterizing i/o performance using the tau performance system, 2011.