Projektbericht

IO-Verhalten und -Effizienz im Klima- und Wettermodell ICON

Florian Ott
Universität Hamburg, B.Sc. Informatik
florian.ott@studium.uni-hamburg.de

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Inhaltsverzeichnis

1 Aufgabenstellung und Motivation

Grundlage dieses Berichts ist ein studentisches Projekt, welches in Kooperation mit dem Deutschen Klimarechenzentrum (DKRZ) durchgeführt wurde. Ziel war es, das Input/Output (IO)-Verhalten des Wetter- und Klimamodells ICON, spezifisch auf dem Supercomputer "Levante" des DKRZ, zu untersuchen. Das Hauptaugenmerk lag dabei auf der Untersuchung der zeitlichen Performance unter verschiedenen Parameterkombinationen.

Dazu wurden verschiedenste Konfigurationen von IO-Modi und Parametern in ICON getestet und analysiert. Die Notwendigkeit der gezielten Betrachtung von IO ergibt sich aus der rapiden Entwicklung der Rechenleistung im Vergleich zu dem eher begrenzten Ausbau von IO-Bandbreite [1]. Dies kann zu einem Bottleneck seitens IO führen. Daher ist die Effizienz von IO im Zusammenspiel mit den eigentlichen Berechnungsvorgängen von zentraler Wichtigkeit, was jedoch ein Verständnis, auch spezifisch anwendungs- und systembezogener Natur, voraussetzt.

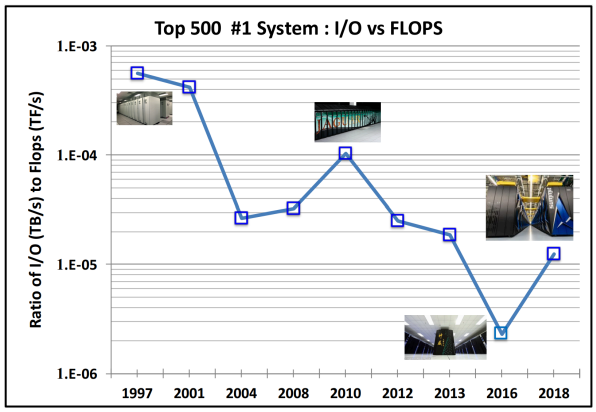


Abb. 1: Compute vs IO Development [1]

Das bisher begrenzte Wissen über IO betont dies besonders. Die Ergebnisse des Projekts sollen in diesem Bericht zusammengefasst und diskutiert werden. Zunächst werden Grundlagen zu ICON und IO in HPC erläutert, bevor die spezifischen Einstellungsmöglichkeiten in ICON und die verschiedenen IO-Modi vorgestellt werden. Darauf folgend werden das für die Messungen zugrundeliegende Experiment vorgestellt und die Ergebnisse diskutiert.

2 Grundlagen

2.1 Das ICON-Modell

ICON (*icosahedral non-hydrostatic*) ist ein Wettermodell, welches in einer Kooperation des Deutschen Wetterdienstes (DWD), dem Max-Planck-Institut für Meteorologie (MPI-M), dem Deutschen Klimarechenzentrum (DKRZ), dem Karlsruhe Institut für Technologie (KIT) und dem Center for Climate Systems Modeling (C2SM) entwickelt wurde. Vom DWD wird es für die operative Wettervorhersage genutzt, vom MPI-M für die Klimaforschung. Der Name leitet sich von der Eigenschaft ab, die Erde als icosahedrales Gitter und die Atmosphäre als nicht-hydrostatisch zu modellieren.

Abb. 2: Ikosahedrales Gitter [2]

Dies soll die Betrachtung feinerer Maschenweiten im Gitter und somit eine genauere Berechnung und Vorhersage von Klima und Wetter ermöglichen [3], [4]. Die horizontale Maschenweite kann so auf bis zu 13km heruntergebrochen werden, bei 120 vertikalen Schichten der Atmosphäre mit einer Höhe bis zu 75km. Die Besonderheit des ikosahedralen Gitters ist die Aufteilung in 20 gleichseitige Dreiecke, welche immer feiner unterteilt werden können. ICON kann auch im Climate Limited-Area Mode (CLM) genutzt werden, wodurch die Fokussierung auf bestimmte Regionen möglich wird. So gibt es ICON-EU mit einer Maschenweite von 6,5km für Europa und ICON-D2 mit einer Maschenweite von 2,1km für Deutschland [4].

Die hohe Auflösung, die mitunter hohe Anzahl an Variablen, die kurze Länge der Zeitintervalle sind Grundlage für die Notwendigkeit sehr feiner Berechnungen, was zu einem hohen Rechenaufwand führt. Dieser wird durch die Nutzung von Supercomputern und Parallelisierung der Berechnungen auf viele Prozessoren durch OpenMP (Open Multi-Processing - innerhalb eines Knotens) und mehrere Rechenknoten (mit Message Passing Interface - MPI) bewältigt. OpenMP wird verwendet, um mehrere Threads in einem Prozess zu starten. Bei der Nutzung von MPI werden mehrere Instanzen von ICON gestartet, wobei jede dieser einen eigenen Teil des Grids berechnet. Hybride Herangehensweisen sind ebenfalls möglich, bei denen sowohl OpenMP als auch MPI genutzt werden [5]. Zur Parallelisierung werden die Zellen des Grids in Blocks aufgeteilt, deren Länge innerhalb der Experimentskripte zu definieren ist. Diese können anschließend als 2D-Array nebeneinandergereiht werden. Weitere Infos folgen in Kapitel 3. Wie genau die IO-Operationen in ICON ablaufen, ist abhängig von der genauen Konfiguration durch die Anwendenden. Die Möglichkeiten dazu sollen im folgenden Kapiteln genauer beleuchtet werden.

2.2 IO in HPC

Die Herausforderung von IO in HPC besteht darin, dass im Kontrast zu herkömmlichem IO in Desktop-Computern Dateisysteme über viele Speichereinheiten verteilt sind, um die hohe Datenmenge zu bewältigen und benötigte Bandbreite zu realisieren. Auf Levante kommt das Dateisystem Lustre zum Einsatz, welches ermöglicht, einen POSIX-konformen Namespace über die Speichereinheiten zu verteilen. Für die Anwendenden wirkt es, als würden sie auf nur einer Speichereinheit arbeiten.

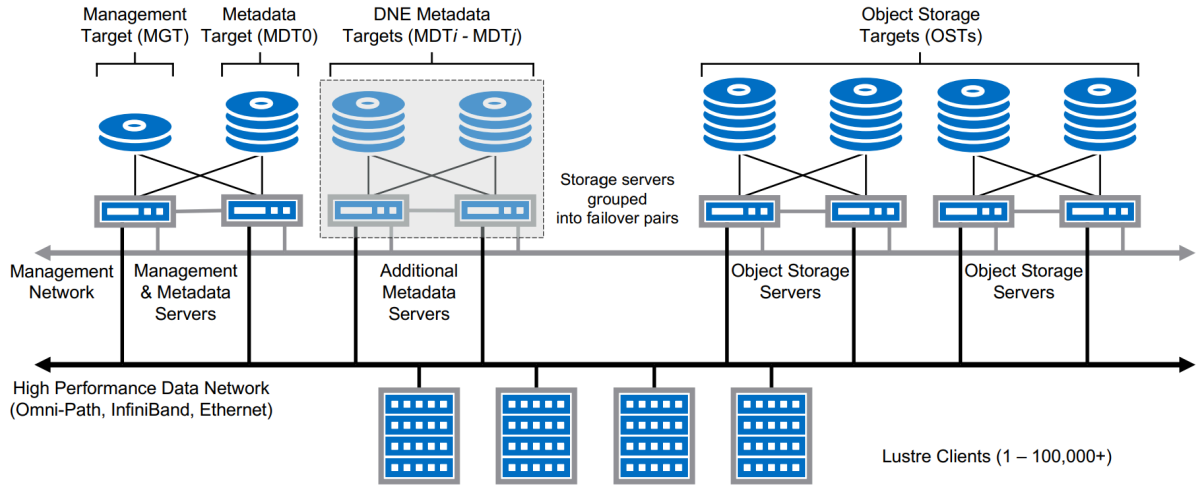


Abb. 3: Lustre-Architektur [6]

Wie in Abb. 3 zu sehen, besteht Lustre aus mehreren Komponenten, die auf verschiedenen Ebenen arbeiten. Auf den MDS (Meta Data Server) werden Metadaten der Dateien gespeichert, welche sich selbst auf OSS (Object Storage Server) befinden. Die Clients sind die Rechner, die auf das Dateisystem zugreifen. Eine weitere Besonderheit ist die Möglichkeit, Dateien in Stripes zu schreiben, also ein File auf mehrere OSTs zu verteilen. Dabei wird die Datei in gleichgroße Teile aufgeteilt und auf die OSTs geschrieben, entsprechend der RAID 0 Konfiguration. Dadurch kann die Bandbreite des Zugriffs auf eine Datei die Bandbreite des Zugriffs auf ein OST übersteigen [6]. Die Kommunikation zwischen den Komponenten erfolgt auf Levante über InfiniBand [7].

Für das Schreiben von Dateien gibt es in HPC verschiedene Optionen.

- Die naive Lösung wäre, den Master Prozess eigenständig und allein die Datei schreiben zu lassen, nachdem Berechnungen über mehrere Prozesse oder Threads parallel durchgeführt wurden - wobei der Output dann ein Bottleneck darstellen würde. IO Operationen erfolgen immer nacheinander, somit **sequentiell**.
- Alternativ kann das Schreiben von Dateien parallelisiert werden, indem jeder Prozess oder Thread eine eigene Datei schreibt (**File Per Process** - FPP). Dies wiederum kann jedoch zu einer hohen Anzahl an Dateien führen, was ebenso die Performance beeinträchtigen kann, da vor allem der Zugriff auf viele kleine Files unter Lustre ineffizient ist. Außerdem muss beachtet werden, dass auch für jeden dieser Files Metadaten gespeichert und gelesen werden müssen, was seinerseits unnötigen Overhead begünstigt.
- Eine weitere Möglichkeit ist das Schreiben in **Shared Files**, bei dem mehrere Prozesse auf eine Datei schreiben. Dies kann die Anzahl der Dateien reduzieren und somit den Zugriff durch User vereinfachen. Allerdings ist die robuste Implementierung dieser Methode schwieriger als FPP.
- Darüber hinaus gibt es **Mischformen** zwischen den verschiedenen Optionen, zum Beispiel dass Shared Files pro Rechenknoten gebündelt werden. So liegt dann quasi eine File Per Node Konfiguration vor, die die Anzahl der Dateien reduziert, jedoch die Anzahl der Prozesse pro Datei erhöht [8].

3 IO Einstellungsmöglichkeiten in ICON

Die Steuerung des IO-Verhaltens in ICON erfolgt über verschiedene Parameter, welche in sogenannten “namelists” (nml) innerhalb der Experimentskripte festgelegt werden. Die genaue Dokumentation dazu ist in [9] zu finden. Zentral sind hier:

- `io_nml`:
  - `restart_write_mode`: Einstellung zum Schreibmodus der Restartfiles (Infos dazu folgen an anderer Stelle),
- `parallel_nml`:
  - `nproma`: definiert Blocklänge, kann auch am Anfang des Skripts als Variable festgelegt werden
  - `num_io_procs`: definiert Anzahl dedizierter IO-Prozesse
  - `num_restart_procs`: definiert Anzahl dedizierter Restart-IO-Prozesse
  - `pio_type`: Festlegen von entweder klassisch asynchronem Schreiben (FPP) oder CDI-PIO (shared file)
- `run_nml`:

  - `output`: Auswahl der Output-Komponenten, vor allem "nml" sei hier zu erwähnen, da so durch die `output_nml`-Namelist genauere Spezifikationen vorgenommen werden können
- `output_nml`:
  - `[domain]_varlist`: definiert die Variablen der entsprechenden Domäne, die geschrieben werden sollen
  - `output_filename`: definiert den Dateinamen
  - `stream_partitions_ml`: Anzahl der Prozesse, welche Output Files des Streams schreiben (insbesondere nützlich, wenn die einzelnen Files so groß sind, dass ein Prozess nicht in der Lage ist, sie zu schreiben, bis bereits ein neues Output File geschrieben werden muss)

Die entscheidendsten Einstellungen sind somit zunächst in der `output_nml` und der `parallel_nml` zu finden. Weitere exemplarisch wichtige Variablen, welche in den Runskripten der einzelnen Experimente zu finden sind, sind:
- `grid_refinement`: definiert die Maschenweite des Gitters
- `start_date`: Startdatum des Experiments
- `end_date`: Enddatum des Experiments
- `output_interval`: Intervall, in dem Ergebnisdaten geschrieben werden
- `file_interval`: Intervall, in dem eine neue Datei eröffnet wird, in welche die Ergebnisdaten geschrieben werden
- `restart_interval`: Intervall, in dem Restart-Dateien geschrieben werden
- `checkpoint_interval`: Intervall, in dem ein Checkpoint erstellt wird

Für jede `output_nml`-Nameliste wird ein File geschrieben, jeweils mit entsprechenden Intervallen, welche in der Namelist selbst definiert werden. Enthalten sind die in `[domain]_varlist` definierten Variablen. Diese Files werden je nach Einstellung der Intervalle in weitere Files unterteilt. Mehrere Outputs können in ein File geschrieben werden, nach erreichen des File Intervals wird eine neue Datei geschrieben. Außerdem gibt es Restart Dateien, die das gesamte Set an Variablen enthalten. Diese werden zu jedem Checkpoint angelegt und können zur Wiederherstellung des Modells genutzt werden. Nach Erreichen des Restart Intervals wird der aktuelle Slurm (job scheduler auf Levante) Job beendet und ein neuer gestartet. Sowohl für Output als auch für Restart gibt es die Möglichkeit, einzelne Prozesse abzustellen, welche sich auf das Schreiben der entsprechenden Dateien konzentrieren.

4 IO Modi in ICON

Die verschiedenen Modi zum Schreiben und Lesen von Dateien in ICON ergeben sich aus den oben genannten Einstellungen.

4.1 Seriell (Master Prozess)

Die einfachste Variante ist, dass der Master Prozess alleine die Datei schreibt, nachdem alle Prozesse ihre Daten sammeln und an den Master schicken (in der Regel Prozess Rank 0). Dies ist jedoch ineffizient, da IO Operationen somit immer sequentiell ablaufen und die Worker auf das Schreiben der Daten warten müssen. Selbiges gilt für das Schreiben der Restart Files. Der Modus ergibt sich aus der Einstellung `num_io_procs = 0`. Die sehr einfache schematische Darstellung der Kommunikation zwischen Worker und IO Prozess ist in Abb. 4 zu sehen.

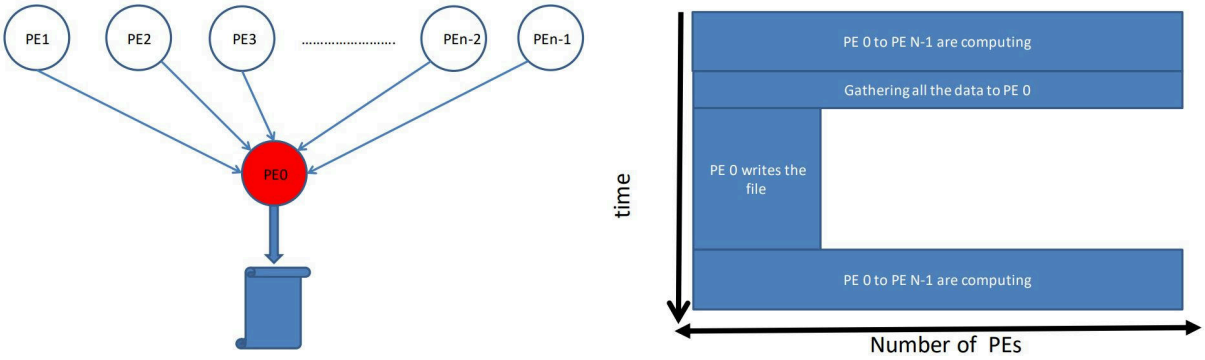


Abb. 4: serielles IO [10]

4.2 Klassisch asynchron (file per process)

Eine weitere Möglichkeit ist das Schreiben von Dateien durch dedizierte IO Prozesse, welcher durch die Einstellung `num_io_procs > 0` aktiviert wird. Die Kommunikation zwischen Worker und IO Prozess ist in Abb. 5 zu sehen. Die Worker schicken ihre Daten an die entsprechenden IO Prozesse, welche den Output in die entsprechenden Files schreiben. Jeder IO Prozess schreibt dabei in ein eigenes File, woher sich der Name “File Per Process” (FPP) ableitet. Dies kann jedoch zu einer hohen Anzahl an Dateien führen, was die Performance beeinträchtigen kann. Der Vorteil ist hier eine zeitlich wesentlich bessere Performance, jedoch ist es nach wie vor möglich, dass eine ungleiche Verteilung von Datenmengen auf Files zu bottlenecks führt. Ebenso werden datenintensive Jobs schnell von verhältnismäßig großen Mengen an Metadaten begleitet [11]. Falls in regelmäßigen Abständen ein sehr großer File geschrieben werden muss und die Last trotzdem nur auf einem IO Prozess liegt, kann dies immer noch Wartezeiten der Worker bedingen (siehe Abb. 6).

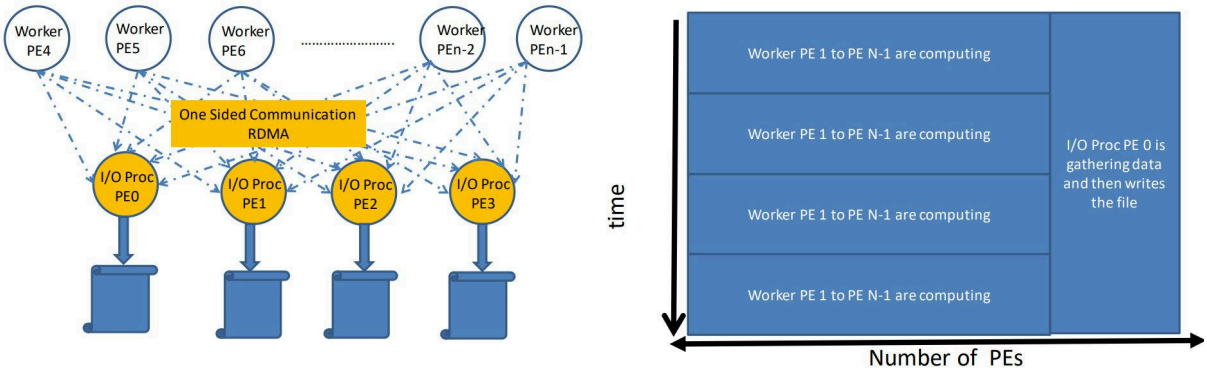


Abb. 5: serielles IO [10]

Eine Option, welche Abhilfe schafft, ist das Partitionieren des Outputs. Dadurch werden große Outputs in kleinere Abschnitte unterteilt, welche jeweils in eigene Files geschrieben werden und somit auf mehrere IO Prozesse verteilt werden können. Die Funktionsweise ist in der rechten Abbildung in Abb. 6 zu beobachten. Dies geschieht mit `stream_partitions_ml` in der `output_nml`. Das Partitionieren wird für jedes Set an Variablen, also jeden Output Stream, individuell festgelegt. Falls keine Unterteilung des Outputs vorgenommen wird, werden alle IO Prozesse, welche über die Anzahl der definierten Output Namelists hinausgehen, idlen.

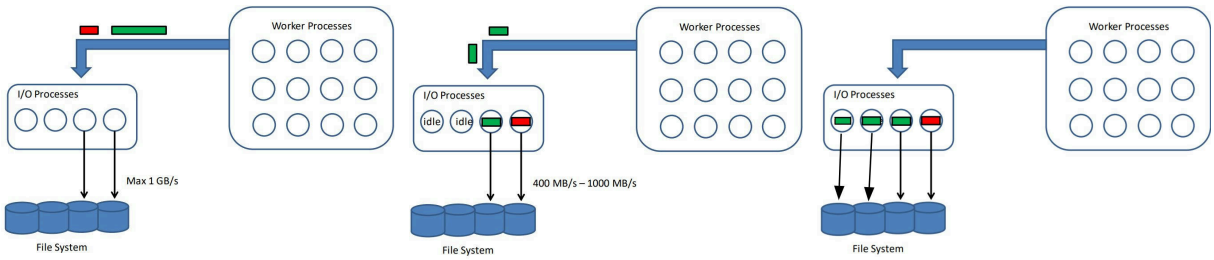


Abb. 6: Stream Partitioning [10]

Parallel kann mit `num_restart_procs` die Anzahl der Prozesse festgelegt werden, die sich auf das Schreiben der Restart Files konzentrieren. Eine Übersicht der Restart Modi, welche in `restart_write_mode` festgelegt werden können:

- `sync`: PE (processing element) #0 schreibt und liest Restart File. Alle anderen PEs warten. `num_restart_procs` MUSS auf 0 gesetzt werden, sonst gibt es Fehlermeldungen.
- `async`: Dedizierte PEs (definiert durch `num_restart_procs > 0`) schreiben Restart Files während Simulation weiterläuft. Lesen geschieht nur durch PE #0.
- `joint procs multifile`: alle PEs schreiben Restart Files in dafür angelegtes Verzeichnis. Dieses Verzeichnis wird als Restart File verwendet. Alle PEs lesen aus diesem Verzeichnis parallel.
- `dedicated procs multifile`: Restart Daten werden von Workern in Buffer der Restart PEs gesendet. Worker PEs führen Berechnungen fort, Restart PEs schreiben die Files. Alle Worker PEs können gleichzeitig lesen
- wird kein Modus spezifiziert, wird bei `num_restart_procs > 0` automatisch `async` gewählt, für einen Wert von 0 `sync` [9].

4.3 CDI-PIO (shared file)

Des Weiteren hat sich die Methode etabliert, mehrere Prozesse auf eine Datei schreiben zu lassen, was als **Shared File IO** bezeichnet wird. Vorteil ist hierbei vor allem die erhöhte Benutzerfreundlichkeit, da bei FPP jeder Prozess eine eigene Datei schreibt, was die Handhabung erschwert. Erreicht wird dies durch das Striping von Files auf mehrere OSTs (siehe Abschnitt 2.2), sodass mehrere PEs in dem File schreiben können, jedoch logisch nur ein File existiert. Im Unterschied dazu wird auch bei dem partitionierten FPP für jede Partition ein komplett eigener File geschrieben. Idealerweise greift immer ein PE auf ein OST zu, bei mehreren PEs reduziert sich die Bandbreite bereits massiv, da zur Sicherung der Datenintegrität nicht mehrere PEs gleichzeitig schreiben können [11]. Da CDI-PIO bislang in ICON eher experimentell implementiert ist, wurde es im Rahmen dieses Projekts nicht weiter betrachtet.

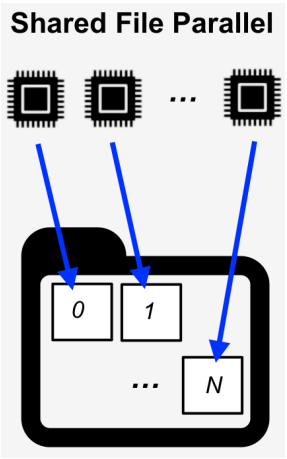


Abb. 7: Shared File IO

5 Messungen

Zentraler Bestandteil der vorliegenden Arbeit sind die Messungen, welche systematisch im Rahmen der Bearbeitung durchgeführt wurden und auf deren Grundlage anschließend eine

grobe Empfehlung zur Wahl der Parameter bei der Konfiguration von ICON gegeben werden soll.

5.1 Methodik

Zur zeitlichen Performanceanalyse wurden verschiedene Konfigurationen von IO-Modi und Parametern in ICON getestet. Die Messungen erfolgten auf dem Supercomputer "Levante" des DKRZ. Das zugrundeliegende Experiment ist das Testexperiment *exp.nh_dcmip_tc_52_r2b5* und eine Version *exp.nh_dcmip_tc_52_r2b5*, d.h. der Tropenzyklon(tc)-Test mit 80km (r2b5) bzw. 40km (r2b6) Maschenweite zum non-hydrostatischen (nh) Dynamical Core Intercomparison Project (dcmip), welches dem Vergleich mehrerer Klimamodelle dient. Das Experiment-Skript wurde auf Empfehlung einer der Projektbetreuer*innen und Angestellten des DKRZ gewählt und wurde mehrfach für die Zwecke dieser Arbeit modifiziert **TODO git rep verlinken**. Unter anderem wurde die Möglichkeit des externen Inputs von Argumenten beim Aufruf des Skriptes geschaffen, um die Handhabbarkeit multipler Aufrufe des Skripts mit verschiedenen Einstellungen zu erleichtern. Die Liste der Output-Variablen, insgesamt 54 verschiedene, wurde größtenteils aus dem bestehenden Skript übernommen und nicht groß abgeändert. Das Experiment berechnet einen Zeitraum von zehn Tagen, auch dies blieb unverändert. Aufgerufen wurde das Skript über ein eigens (rudimentär, da pragmatisch) geschriebenes Pythonskript (*mult_run.py*), in welchem Einstellungen für das aufzurufende Skript vorgenommen werden konnten, welche am Ende automatisch in das Log übertragen wurden. Ein weiterer Vorteil war die Möglichkeit, die Logs direkt in den entsprechenden Ordner der zugehörigen Experiment-Einstellungen zu verschieben, was die manuelle Zuordnung der Logs zu den Einstellungen erleichterte. Nach ersten Tests erfolgte die Festlegung auf Parameterkombinationen, wobei zunächst IO und Restart bzw. Checkpoints separat betrachtet wurden. Zu dem Restart sei zu erwähnen, dass nur auf das Schreiben von Checkpoints eingegangen wird, nicht auf den eigentlichen Restart, da hierfür ein neuer Slurm-Job gestartet wird und die Erhebung der Daten für die Messungen dadurch für den Rahmen dieses Projekts zu aufwändig gewesen wäre. Dementsprechend ist dies ein Bereich, der sicherlich in anderen Projekten noch einmal genauer untersucht werden könnte.

Für die Untersuchung bezüglich des IO-Verhaltens wurden folgende Parameterkombinationen getestet:

- 80km Grid, 4min Output Intervall, 1 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse
- 80km Grid, 4min Output Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse
- 80km Grid, 12min Output Intervall, 1 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse
- 80km Grid, 12min Output Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse
- 40km Grid, 12min Output Intervall, 1 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse
- 40km Grid, 12min Output Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse
- 40km Grid, 12min Output Intervall, 8 Knoten, davon je 0, 1, 2, 4, 8, 16, 32 IO Prozesse

Dabei wurde die Anzahl der IO Prozesse schrittweise erhöht, um die Auswirkungen auf die Performance zu untersuchen. Für alle Durchläufe wurde das Intervall des Schreibens von Checkpoints auf 12 Tage gesetzt, sodass keine geschrieben wurden und das Ergebnis nicht beeinflussen konnten.

Für die Untersuchung bezüglich des Schreibens von Checkpoints wurden folgende Parameterkombinationen getestet:

- 80km Grid, 12m Checkpoint Intervall, 1 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 80km Grid, 12m Checkpoint Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 80km Grid, 12m Checkpoint Intervall, 4 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 80km Grid, 12h Checkpoint Intervall, 1 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 80km Grid, 12h Checkpoint Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 80km Grid, 12h Checkpoint Intervall, 2 Knoten, Modus "joint procs multifile"
- 80km Grid, 12h Checkpoint Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse, Modus "dedicated procs multifile"
- 40km Grid, 12h Checkpoint Intervall, 1 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 40km Grid, 12h Checkpoint Intervall, 2 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse

Das Output Intervall wurde bei diesen Messungen auf 12 Tage gesetzt und überschritt somit den Experimentzeitraum, analog zum Vorgehen bei den IO Messungen. Soweit nicht anders angegeben war der angewendete Restart Modus immer "async", wobei bei 0 Restart Prozessen "sync" gewählt werden muss, da es sonst zu Fehlermeldungen kommt.

Um zu sehen, wie sich die Einstellungen von IO und Restart aufeinander auswirken und sich zusammen verhalten wurden abschließend verschiedene Kombinationen getestet:

- 80km Grid, 12m Output Intervall, 12h Checkpoint Intervall, 1 Knoten, 0 IO, je 1, 4, 8 Restart Prozesse
- 80km Grid, 12m Output Intervall, 12h Checkpoint Intervall, 1 Knoten, 1 IO, je 1, 4, 8 Restart Prozesse
- 80km Grid, 12m Output Intervall, 12h Checkpoint Intervall, 1 Knoten, 2 IO, je 1, 4, 8 Restart Prozesse
- 80km Grid, 12m Output Intervall, 12h Checkpoint Intervall, 1 Knoten, 4 IO, je 1, 4, 8 Restart Prozesse
- 80km Grid, 12m Output Intervall, 12h Checkpoint Intervall, 1 Knoten, 8 IO, je 1, 4, 8 Restart Prozesse

TODO ggf 12m12m Messungen machen

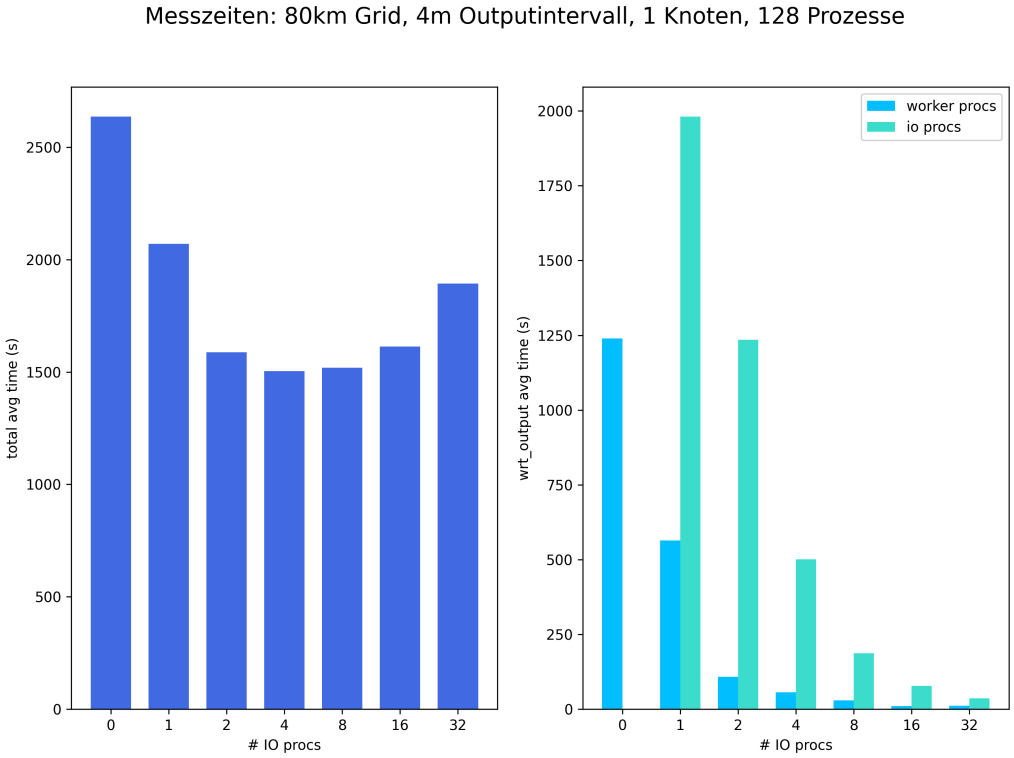
Die Auswertung erfolgt größtenteils auf Basis der Timer Reports innerhalb der Logfiles, welche anschließend automatisiert per Python-Skript **TODO quelle skript** ausgelesen und mit matplotlib visualisiert wurden. Außerdem wurde Darshan genutzt, um das IO-Verhalten der Prozesse während der Laufzeit zu analysieren. Darshan ist ein skalierbares Tool zur Auswertung von IO-Operationen, welches minimale Auswirkungen auf die Performance hat und einfach zur Runtime im Jobskript aktiviert werden kann. Dazu reicht das Einfügen von `LD_PRELOAD=/pfad/zu/darshan/lib/libdarshan.so` in den Header des Skripts. Während der Durchführung des Experiments kam es zu Problemen bei der Erstellung der Darshan Logs, weshalb nach einiger Recherche zur Ursache dessen die Loghints des Tools mit `DARSHAN_LOGHINTS=` deaktiviert wurden. Zur Übersicht zu genutzter Rechenkapazität und Bandwidth erfolgte außerdem eine grobe Analyse der Logs aus ClusterCockpit, welches Daten aus allen laufenden Jobs aus Levante zusammenträgt und clusterseitig eine Analyse ermöglicht.

5.2 Ergebnisse

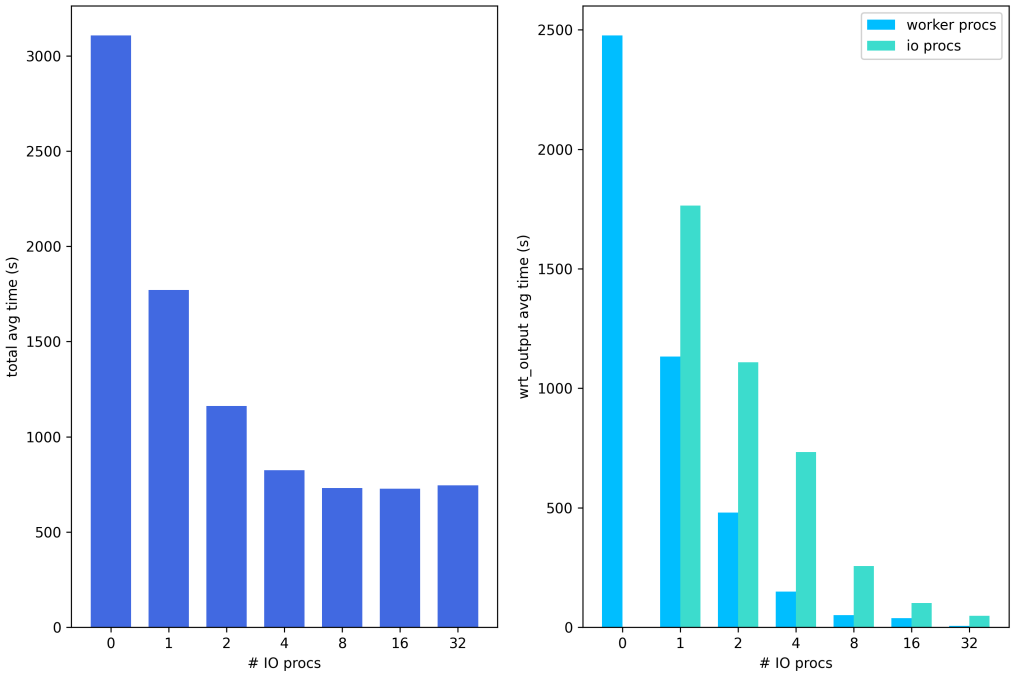
5.2.1 Allgemein

5.2.2 IO

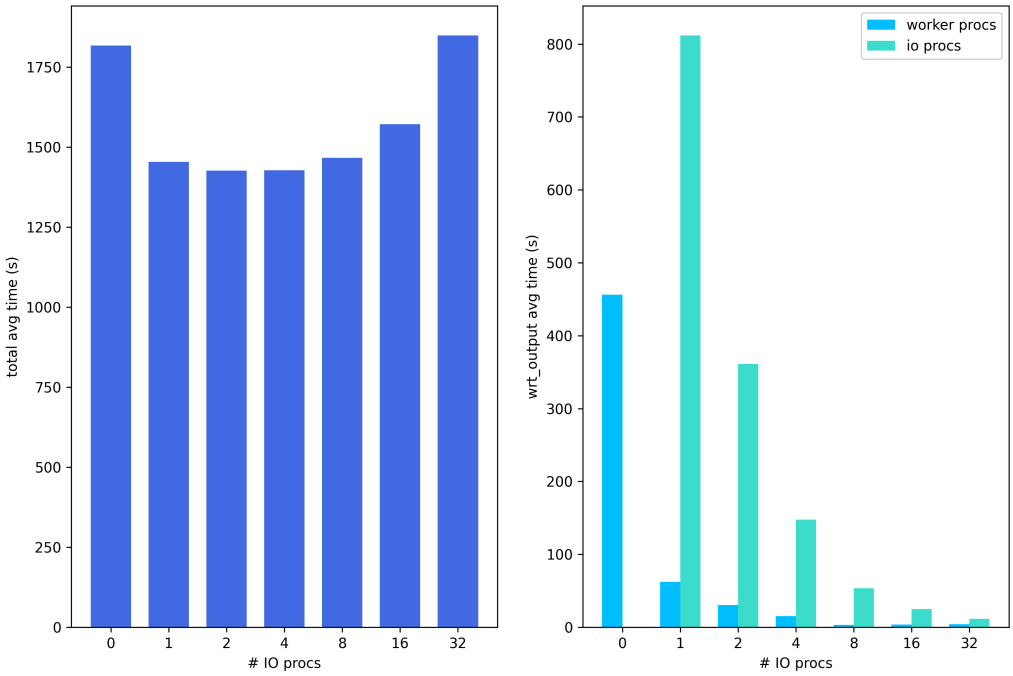
Wie zuvor beschrieben, wurde bei vorab durchgeführten Testläufen deutlich, dass die Anzahl der Streams, in die ein Output partitioniert wird, optimalerweise auf die Anzahl der IO Prozesse abgestimmt sein sollte. Dies wurde in den Messungen berücksichtigt, indem die Anzahl der Streams auf die Anzahl der IO Prozesse gesetzt wurde.

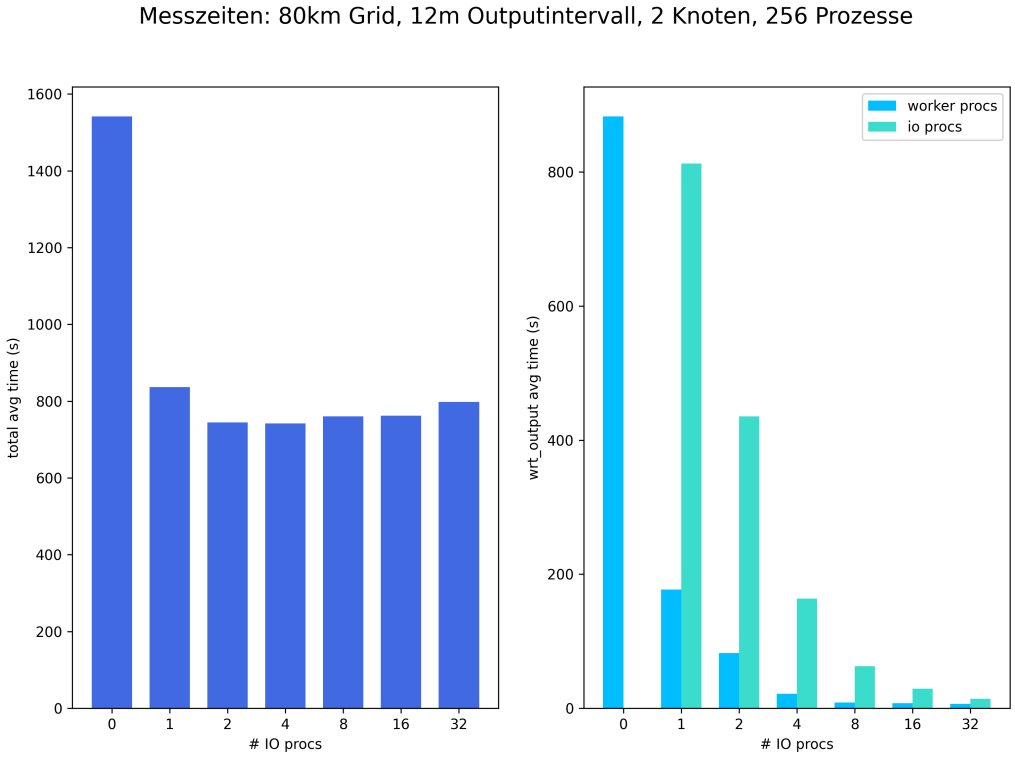


Messzeiten: 80km Grid, 4m Outputintervall, 2 Knoten, 256 Prozesse



Messzeiten: 80km Grid, 12m Outputintervall, 1 Knoten, 128 Prozesse





5.2.3 Checkpoints

5.2.4 Kombiniert

6 Interpretation und Diskussion

7 Notizen

- Lustre FPL, Striping für Benchmarking? Details zu Levante?
- Tools: Darshan
- Restart Write Modi
- siehe Tutorial S. 151 -> Datenstruktur Arrays Grid Blocks
- Kommunikation zwischen worker und IO/restart über mehrere Knoten - wie verteilen sich die Prozesse auf die Knoten?

8 Anmerkungen

Der vorliegende Bericht wurde mit einem modifizierten Template für typst angefertigt [12].

Bibliography

[1] R. J. Zamora, “A Brief Introduction to HPC I/O.” Accessed: Jun. 15, 2024. [Online]. Available: https://indico.fnal.gov/event/18091/contributions/45649/attachments/28369/35073/HEPIOworkshop-Zamora.pdf

[2] “Icon Icosahedral Grid Symbol.” Accessed: Jun. 13, 2024. [Online]. Available: https://wisskomm.social/system/media_attachments/files/111/851/268/526/771/323/original/743ef520429e4901.png

[3] “ICON (Wettervorhersagemodell).” Accessed: Jun. 13, 2024. [Online]. Available: https://de.wikipedia.org/wiki/ICON_(Wettervorhersagemodell)

[4] “Numerische Modellvorhersagedaten.” Accessed: Jun. 13, 2024. [Online]. Available: https://www.dwd.de/DE/leistungen/modellvorhersagedaten/modellvorhersagedaten.html

[5] “Working with the ICON Model.” Accessed: Jun. 17, 2024. [Online]. Available: https://code.mpimet.mpg.de/attachments/download/19568/ICON_tutorial_2019.pdf

[6] “Introduction to Lustre Architecture.” Accessed: Jun. 15, 2024. [Online]. Available: https://wiki.lustre.org/images/6/64/LustreArchitecture-v4.pdf

[7] “Levante HPC System.” Accessed: Jun. 15, 2024. [Online]. Available: https://docs.dkrz.de/doc/levante/index.html

[8] F. Wang and S. Oral, “Introduction to HPC Parallel I/O.” Accessed: Jun. 15, 2024. [Online]. Available: https://www.emsl.pnnl.gov/MSC/UserGuide/_downloads/db39408d8695c108b8e95ae64aa3acd4/2016_HPC_IO_Intro.pdf

[9] “ICON Namelist Overview.” Accessed: Jun. 15, 2024. [Online]. Available: https://gitlab.dkrz.de/icon/icon-model/-/blob/release-2024.01-public/doc/Namelist_overview.pdf

[10] P. Adamidis, “IO in Climate Modelling.” Accessed: Jun. 21, 2024. [Online]. Available: https://events.dkrz.de/event/62/contributions/393/attachments/83/163/IO-in-Climate_Modelling.pdf

[11] P. Farrell, “Shared File Performance Improvements.” [Online]. Available: https://wiki.lustre.org/images/f/f9/Shared-File-Performance-in-Lustre_Farrell.pdf

[12] “simple-typst-thesis.” Accessed: Jun. 15, 2024. [Online]. Available: https://github.com/zagoli/simple-typst-thesis/tree/main?tab=Apache-2.0-1-ov-file