

# *Projektbericht*

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

**Florian Ott, Matr.-Nr.: 7512841**

Universität Hamburg, B.Sc. Informatik

florian.ott@studium.uni-hamburg.de



### **ABSTRACT**

ICON ist ein Wetter- und Klimamodell, welches im deutschsprachigen Raum zur operativen Wettervorhersage und Klimaforschung genutzt wird. Die vorliegende Arbeit beschäftigt sich mit dem IO-Verhalten des Modells auf dem Supercomputer Levante des Deutschen Klimarechenzentrums (DKRZ). Ziel ist es, die Performance unter verschiedenen IO-Modi und Parametern zu untersuchen. Dazu werden verschiedene Konfigurationen getestet und die Ergebnisse diskutiert. Die Ergebnisse zeigen, dass die IO-Performance stark von der Anzahl der IO-Prozesse und der Partitionierung des Outputs abhängt. Bezuglich des zu wählenden Restart-Modus ist der “joint procs multifile” Modus der aktuell am unkompliziertesten anwendbare bei gleichzeitig guter Performanz. Die Ergebnisse der Arbeit zeigen jedoch auch Notwendigkeiten der weiteren Untersuchung auf, insbesondere zu Parametern zu IO des eigentlichen Ergebnisoutputs.

# Inhaltsverzeichnis

1 Aufgabenstellung und Motivation .....	1
2 Grundlagen .....	1
2.1 Das ICON-Modell .....	1
2.2 IO in HPC .....	2
3 IO Einstellungsmöglichkeiten in ICON .....	3
4 IO Modi in ICON .....	4
4.1 Seriell (Master Prozess) .....	4
4.2 Klassisch asynchron (file per process) .....	5
4.3 CDI-PIO (shared file) .....	6
5 Messungen .....	7
5.1 Methodik .....	7
5.2 Ergebnisse .....	8
5.2.1 IO .....	8
5.2.2 Checkpoints .....	14
6 Interpretation und Diskussion .....	22
6.1 Restart/Checkpoint-IO .....	23
6.2 Output-IO .....	23
7 Herausforderungen und Limitationen .....	24
8 Versicherung an Eides Statt .....	25
Bibliography .....	25

## 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.

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.

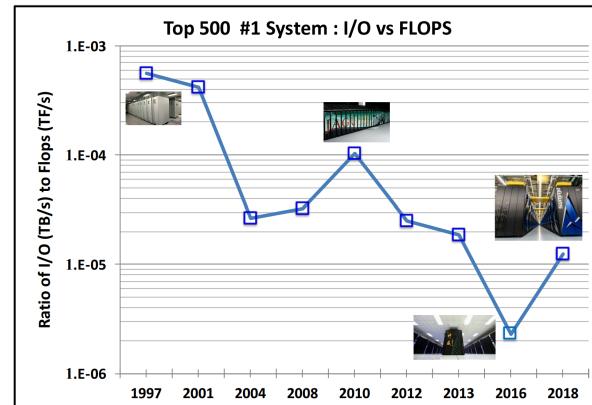


Abb. 1: Compute vs IO Development [1]

## 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 ikosahedrales 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 nebeneinander gereiht 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.

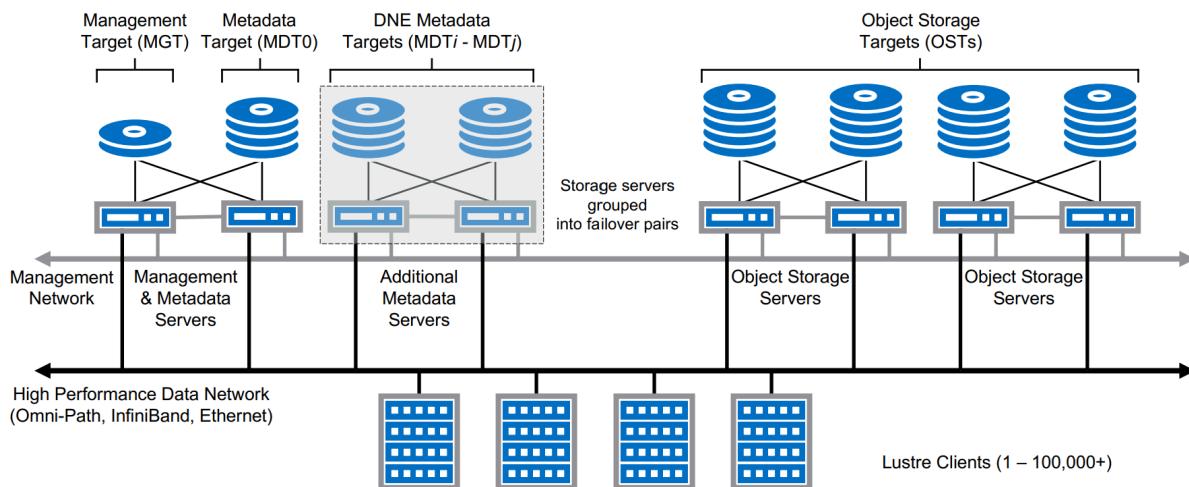


Abb. 3: Lustre-Architektur [6]

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**-Nameliiste 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.

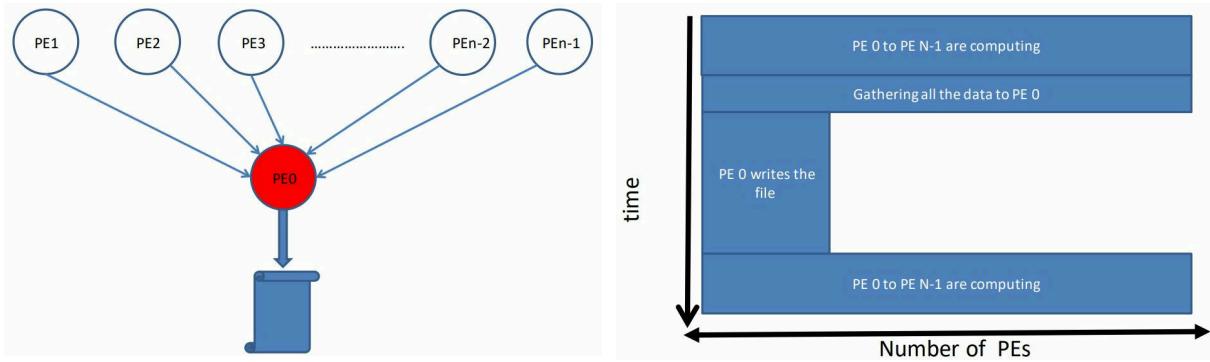


Abb. 4: serielles IO [10]

## 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).

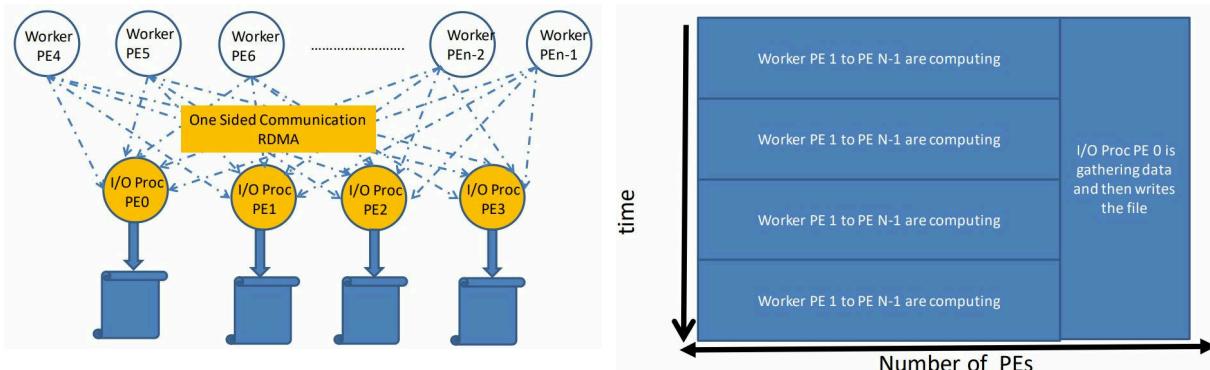


Abb. 5: serielles IO [10]

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. In ICON funktioniert dies, indem die Outputzeit einen Offset bekommt. Ist ein Intervall von 4m für das Schreiben von Outputs vorgesehen, so wird bei einer Partitionierung mit 2 Streams ein Prozess nach 4, 12, 20, ... Minuten schreiben und der andere nach 8, 16, 24, ... Folglich entstehen jedoch mehrere kleine Dateien, statt einer großen zusammenhängende, was die Nutzer\*innenfreundlichkeit erschwert. 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.

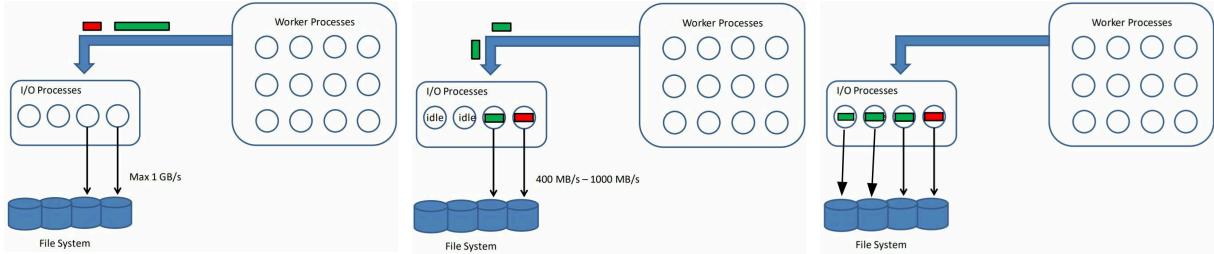


Abb. 6: Stream Partitioning [10]

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 Benutzer\*innenfreundlichkeit, 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.

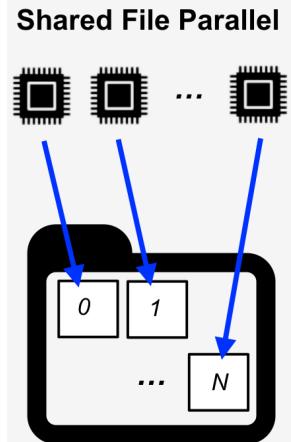


Abb. 7: Shared File IO

## 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 [12]. 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 Python-Skript (*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, 4 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, 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, 12m Checkpoint Intervall, 1, 2, 4 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 80km Grid, 12m Checkpoint Intervall, 1, 2, 4 Knoten, Modus “joint procs multifile”
- 80km Grid, 12m Checkpoint Intervall, 1, 2, 4 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse, Modus “dedicated procs multifile”
- 40km Grid, 12h Checkpoint Intervall, 1, 2, 16 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse
- 40km Grid, 12h Checkpoint Intervall, 16 Knoten, davon je 0, 1, 2, 4, 8 Restart Prozesse, Modus “dedicated procs multifile”
- 40km Grid, 12m Checkpoint Intervall, 16 Knoten, Modus “joint procs multifile”

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.

Die Auswertung erfolgt größtenteils auf Basis der Timer Reports innerhalb der Logfiles, welche anschließend automatisiert per Python-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 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. Jeder zu schreibende Output ist für das Set an Variablen im Experiment auf einem 80km Grid etwa 254MB groß, auf einem 40km Grid etwa 1020MB - wie zu erwarten also eine knappe Vervierfachung bei einer Halbierung der Maschenweite. Die genaue Größe variiert je nach Anzahl der dezidierten Prozesse und Streams aufgrund anfallender Metadaten. Nachstehend abgebildet sind die Ergebnisse der Messungen auf dem 80km-Grid je mit 4min und 12min Output Intervall sowie 1 und 2 Knoten.

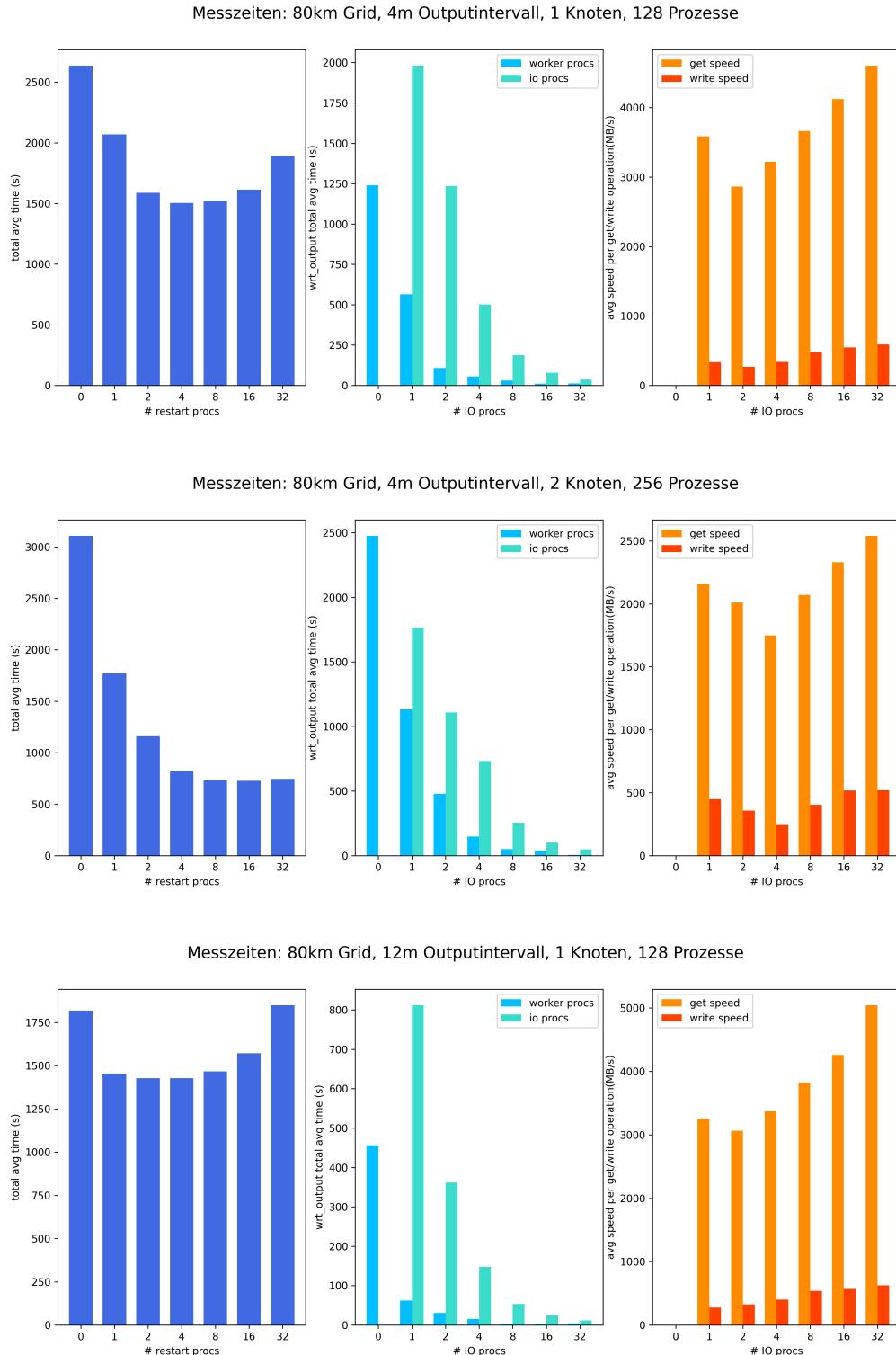
Die Anzahl der IO Prozesse wurde schrittweise von 0 bis 32 erhöht. Eine kurze Erklärung zu den hier abgebildeten Graphen von links nach rechts:

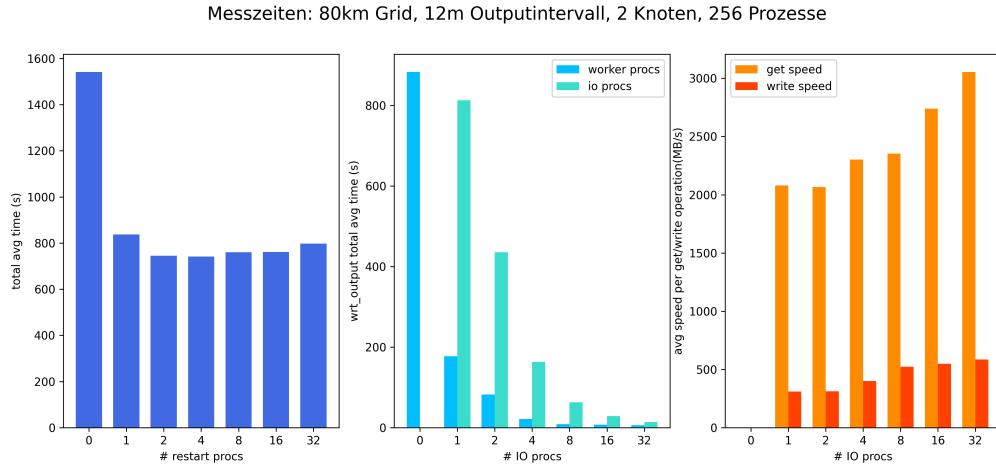
- der erste Graph bildet die über alle Nicht-IO-Prozesse ermittelte “total”-Laufzeit, welche sich insgesamt mit nur sehr leichten Abweichungen mit der Experimentlaufzeit deckt, ab
- der zweite Graph zeigt die kumulierte Zeit, welche sowohl worker als auch IO-Prozesse im Durchschnitt für das Schreiben der Outputs benötigen
- der dritte Graph stellt die Geschwindigkeit des Schreibens der Outputdaten dar, einmal im Sinne der Übertragung an die IO Prozesse und einmal im Sinne des Schreibens auf die Ziel-Datei

Alle diese Daten wurden aus den Logs der Experimente extrahiert - leider wurden jedoch für die Durchläufe ohne spezifische IO PEs in den Logs keine Daten zu den Throughputs der IO Prozesse gespeichert, weshalb die Graphen an dieser Stelle leer blieben. Diese werden wenn möglich durch Erklärungen aus dem ClusterCockpit ergänzt.

Deutlich wird, dass das asynchrone IO, d.h. `num_io_procs > 0`, die Performance in jeder Konfiguration deutlich verbessert. Die Performancesteigerung ist dabei nicht linear, sondern zeigt eine Art Sättigungseffekt, der sich je nach konkreten Einstellungen ab einer bestimmten Anzahl von IO-Prozessen einstellt. Diese Beobachtung leitet sich auch aus den jeweils mittigen Graphen ab, in welchen die Zeit, welche sowohl worker als auch IO Procs mit dem Output verbringen, teilweise exponentiell abnimmt, entsprechend der sich verdoppelnden Anzahl der IO Prozesse. Somit wird der Zeitgewinn bei Hinzufügen weiterer IO Procs weiter gegen 0 gehen. Die Gesamtgröße des Outputs ist bei 12 Minuten Intervall knapp über 280 GiB groß, bei 4 Minuten sind es um die 860GiB. Zu sehen ist, dass die Laufzeit des Experiments auf 2 Knoten bei einem 4m-Intervall und synchronem IO im Vergleich zu der Version auf nur 1 Knoten zunächst ansteigt, was sich durch das sequentielle Schreiben und den Overhead erklären lässt, der sich durch die Kommunikation zwischen den beiden Knoten ergibt. Ein deutlicher Speedup ist erst ab mindestens zwei dezidierten IO PEs zu erkennen. Außerdem zeigt sich, dass aufgrund des steigenden Anteils der IO-Procs, welche nicht für die eigentliche Berechnung der Daten zur Verfügung stehen, an den Gesamtprozessen nach einem ersten Abfall der Laufzeit diese bei einer zu hohen Anzahl an IO Prozessen wieder ansteigt. Ebenfalls den Erwartungen entsprechend verdoppelt sich die write Zeit (mittiger Graph) bei Verdopplung der Rechenknoten bei sequentiell IO, da dementsprechend doppelt so schnell Outputs geschrieben werden müssen, der Masterprozess aber immer noch das bottleneck darstellt. Gibt es jedoch IO Prozesse, so bleibt die Größendimension ihrer write Zeit pro Prozess etwa dieselbe, auch bei Verdopplung der Knoten. Interessant ist weiter, dass die Write-Geschwindigkeit (rechter Graph) immer einigermaßen konstant bleibt, die Übertragungszeit zu den IO Prozessen jedoch bei Erhöhung der Anzahl an Knoten sinkt - entsprechend der Limitation von Datenübertragung zwischen den Knoten. Weshalb sich die Übertragungszeit jedoch bei der zunehmende Zahl an IO Procs erhöht, ist nicht komplett klar. Möglicherweise wird der Transfer der Daten an die IO Procs dadurch behindert, dass, je mehr Prozesse parallel mit einem anderen Prozess kommunizieren wollen, die Bandbreite für jeden einzelnen sinkt. Dazu wäre ein noch genauerer Blick in den Quellcode von ICON oder auf Tools wie Score-P notwendig, um die zugrundeliegende Kommunikation zwischen Prozessen zu verstehen, die jedoch den Rahmen dieser Arbeit sprengen würde. Jedoch bleibt zu bemerken, dass die write-Geschwindigkeit, welche in den Logs angegeben wird, rechnerisch nicht zu der daneben angegebenen “time

“write” passt, sondern prinzipiell doppelt so hoch sein müsste. Vermutlich müssten also die Balken in den Diagrammen für die write-Geschwindigkeit halbiert verdoppelt werden, um die tatsächliche Geschwindigkeit darzustellen. Die get-Geschwindigkeit passt hingegen zu der angegebenen “time get” und ist somit korrekt. Auf einem Rechenknoten stehen 256 CPU-Cores zur Verfügung, wir rechnen jedoch nur auf 128 Prozessen, d.h. 2 Cores pro Prozess, eventuell ist das eine mögliche Begründung.





Ähnlich bilden sich die Beobachtungen ab, die man aus den Aufzeichnungen des ClusterCockpits ziehen kann. Die GFLOPs/s beginnen bei der 4 Knoten, 80km, 4min Version mit 0 IO Procs bei etwa 95 und steigern sich bis zu 4 IO Procs auf etwa 170, bevor sie langsam wieder abnehmen. Ähnlich verhält es sich bei der InfiniBand und Lustre Bandbreite (0,35 auf 1,2 GB/s und 0,35 auf 0,6 GB/s, respektive). Spannend ist, dass nur bei 0 IO Procs die Lustre BW genau der IB BW entspricht, ab 1 IO und aufsteigend ist die IB BW immer doppelt so hoch wie die Lustre BW, vermutlich bedingt durch die Übertragung der worker an die IO Prozesse. Nur ein Knoten trägt zu der Lustre BW bei, es ist also ersichtlich, dass nur einer von zwei Knoten, nämlich eben jener, auf dem die IO Prozesse laufen, tatsächlich Files schreibt. Der andere Knoten hat dafür eine höhere Rechenleistung (mehr GFLOP/s). Das primär interessante an den Darshan Reports ist die IO Performance bzgl. der POSIX Operationen. Laut Darshan nimmt in allen Konfigurationen die Schreibgeschwindigkeit mit steigender Anzahl an IO Procs insgesamt zu, jedoch scheint die Performance mit 1 IO Proc im Vergleich zum synchronen IO langsamer zu sein. Hauptprädiktor für die Geschwindigkeit ist die Anzahl der IO Prozesse, auch die Zahl der Knoten beeinflusst diese nicht allzu sehr. Bei 32 IO Procs beträgt sie bis zu 36000 MiB/s.

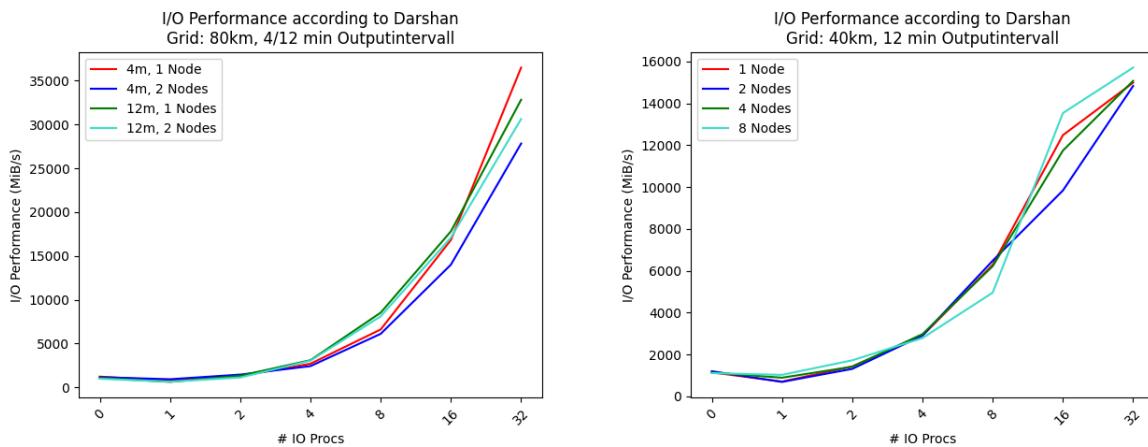


Abb. 12: Darshan I/O Performance auf 80km und 40km Grid

Die nächsten zu betrachtenden Konfigurationen des Experiments basieren alle auf dem Grid mit 40km Maschenweite, mit einem Outputintervall von 12 Minuten und bei steigender Knotenzahl (1, 2, 4, 8). Auch hier verlaufen der Großteil der Beobachtungen erwartungsgemäß.

Die Entwicklungen der Laufzeiten bei Erhöhung der Zahl der IO Prozesse ist entsprechend der zunehmenden Zahl an Rechenknoten und damit -leistung deutlich sichtbarer desto mehr Knoten verwendet werden, da die Frequenz von IO Operationen durch höhere Rechengeschwindigkeit zunimmt. Bei 1 Knoten wird schon ab 1 IO Proc ein Plateau erreicht, bei 2 Knoten ab 2 IO Proc, bei 4 Knoten ab 4 Proc und bei 8 Knoten ist ab 16 Knoten keine große Verbesserung der Laufzeit mehr erwartbar. Spannenderweise nimmt die Laufzeit ohne dedizierten IO Prozess bei Erhöhung der Knotenzahl erst ab und dann wieder leicht zu. Schaut man auf die durchschnittliche Zeit, die Prozesse mit dem Schreiben beschäftigt sind, wird auch sehr schnell die Bedeutung von IO als möglichem Bottleneck deutlich, da sich insbesondere von 4 auf 8 Knoten die Zeit fast verdoppelt. Schaut man sich die Write Zeit eines einzelnen IO Prozesses an, erkennt man, dass diese sich immer um die 3000 Sekunden bewegt, da sich weder die Anzahl der Outputs, noch die Größe der zu schreibenden Daten verändert. Das Schreiben braucht den Prozess immer konstant lang, nur müssen die worker PEs kumuliert immer länger auf das IO warten. Es scheint also sinnvoll, sich Gedanken darüber zu machen, wie hoch der benötigte Throughput des durchzuführenden Experiments ist und wie viele Prozesse benötigt werden, die sich explizit nur um das IO kümmern. Außerdem lässt sich feststellen, dass bei effizienter Konfiguration der Anzahl der IO Prozesse jedesmal eine Halbierung der Laufzeit erreicht werden kann, wenn man die Rechenleistung verdoppelt. Ein Indikator dafür könnte das Verhältnis zwischen wrt\_output avg time der worker und derselben Zeit der IO Procs sein - wenn erstere mehr als die Hälfte der zweiteren beträgt, lohnt sich eine weitere Erhöhung der IO Prozesse. Auch in diesen Messungen kann wieder beobachtet werden, dass zumindest für 1 und 2 Knoten die Geschwindigkeit bei dem Übertragen der Daten auf die IO Prozesse bei steigender Zahl an IO Procs zunimmt, während die write-Geschwindigkeit begrenzt auf maximal 500 MB/s scheint.

Die Beobachtungen aus Darshan zu den Messungen auf dem 80km Grid lassen sich vom Verlauf her übertragen, jedoch erreicht die IO Performance hier nur bis zu 15000 MiB/s bei 32 IO Prozessen. Rechts abgebildet ist eine Heatmap aus dem entsprechenden Darshan Log abgebildet, aus der sichtbar wird, dass 32 der 128 Prozesse regelmäßig Daten schreiben, jedoch häufiger in einem Leerlauf landen. Optimal wäre, wenn die IO Prozesse durchgängig laufen, da sonst unnötiges Overhead entsteht. Die markanteste Auffälligkeit bei den Logs von ClusterCockpit ist, dass mit zunehmenden Knoten die Lustre BW zunimmt.

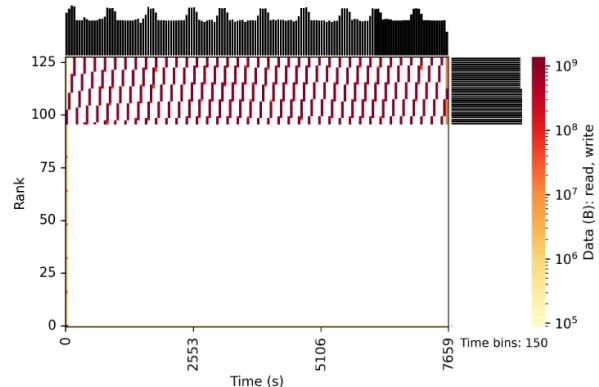
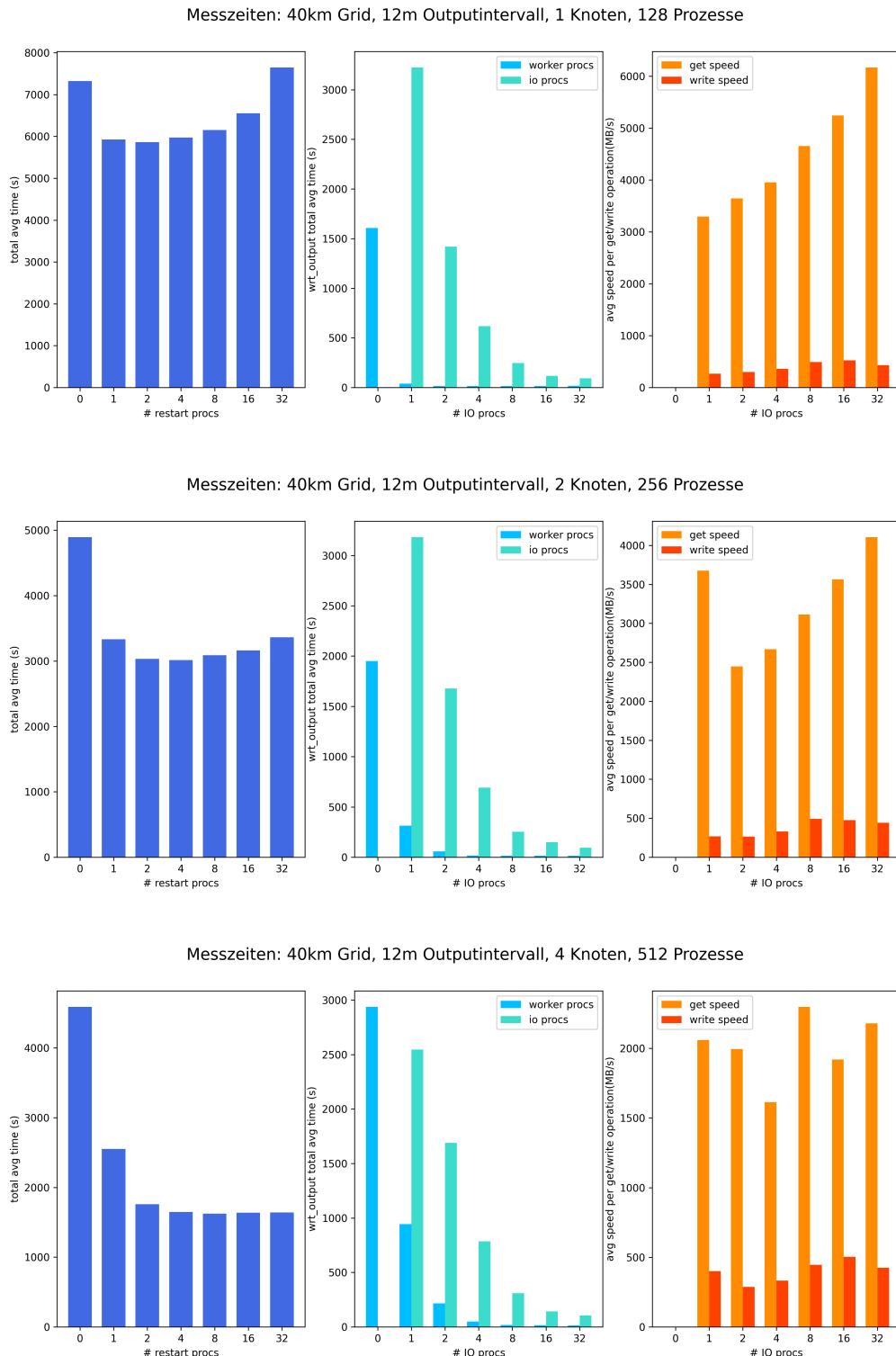
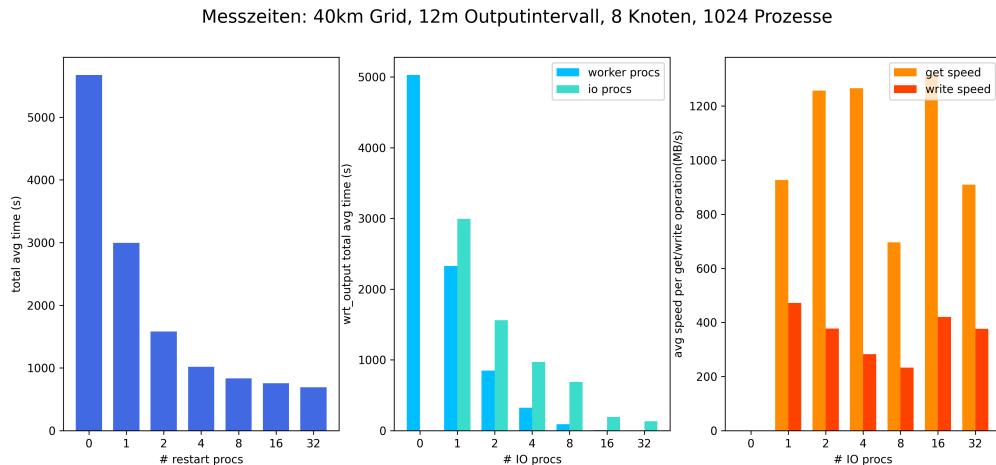


Abb. 13: Darshan Heatmap 40km Grid, 1 Knoten, 32 IO Procs



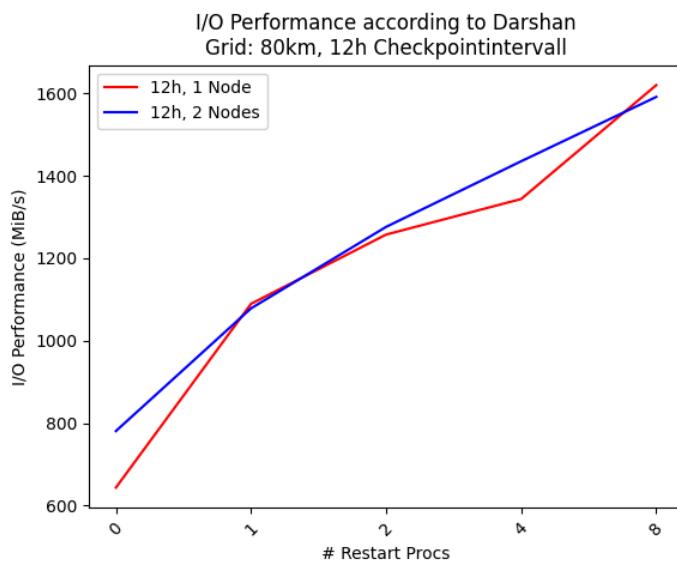


Ein Problem kommt durch die Abweichung der Daten zwischen Darshan, CC und den Logs auf. Als Beispiel sei der Fall mit 4 Minuten Intervall, 80km Maschenweite, 1 Knoten und 8 IO Procs genannt. Darshan registriert im POSIX Modul 856,59 GiB geschrieben Dateien bei einer durchschnittlichen Geschwindigkeit von 6612,29 MiB/s. Aus den Logs ergeben sich kumuliert 854,26 GiB geschriebene Daten bei einer durchschnittlichen Schreibgeschwindigkeit von 480,93 MB/s  $\approx$  458,66 MiB/s pro Schreibvorgang. Selbst bei der Annahme, dass alle 8 IO PEs gleichzeitig geschrieben haben, ergeben sich so nur  $\sim$ 3669 MiB/s an Throughput. Gehen wir von etwa 855 GiB Gesamtoutput aus, ließen sich mit der Einschätzung aus Darshan so etwa 129 Sekunden Schreibzeit errechnen, mit dem Ergebnis aus den kumulierten Outputzeilen der Logs 233 Sekunden, bei um die 25 Minuten Gesamlaufzeit des Experiments. Beides liegt weit von den durchschnittlich  $\sim$ 186 Sekunden entfernt, die jeder IO Prozess laut Timer Report mit wrt\_output verbracht hat. Zieht man nun noch ClusterCockpit hinzu, beobachtet man eine Spitzenauslastung von etwa 1,25 GB/s für die Bandbreite auf InfiniBand und 0,6 GB/s Bandbreite aus Lustre selbst, wobei auch keiner der Werte mit den bisherigen wirklich zusammenpasst. Dementsprechend ist hier gut ein Trend erkennbar, eine tiefergehende Analyse und Vorhersage wird jedoch erschwert. Weiter dazu in Kapitel 6.

### 5.2.2 Checkpoints

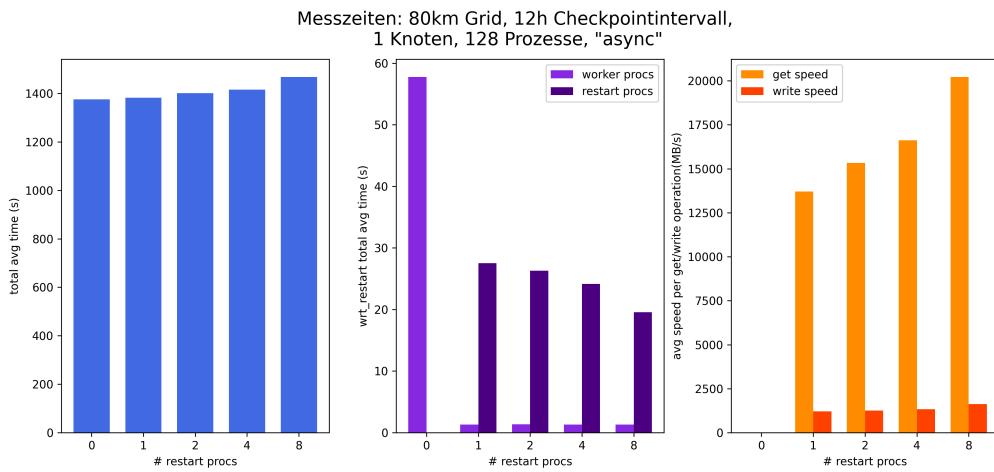
Hier ist nochmal darauf hinzuweisen, dass nur das Schreiben von Checkpoints betrachtet wird, nicht das eigentliche Restarten, allerdings wird letzteres auch durch die Einstellung der Restart Procs beeinflusst, nicht durch die IO Procs. Die Messungen zu den Checkpoints zeigen ein etwa ähnliches Bild wie die zu den IO Prozessen. Weiter unten werden alle entsprechenden Graphiken aufgelistet, welche ähnlich zu lesen sind wie die obenstehenden. Ein Unterschied ist hier, dass zwischen verschiedenen Restart-Modi (siehe Section 4.2) unterschieden wird. In den Graphen für die Messungen des “dedicated procs multifile” Modus ist neben den Balken für unterschiedlichen Runs auch noch die Messung zum “joint procs multifile” Modus inkludiert, da dieser, wie in der Beschreibung schon spezifiziert, nicht über dezidierte IO Prozesse läuft, sondern jeder Prozess als IO Prozess fungiert und somit die Einstellung der Anzahl der Restart Procs wegfällt. Die Daten zu den Runs mit 0 Restart Prozessen bei den “async” und “dedicated” Modi sind dieselben, da bei beiden der Run eigentlich mit dem “sync” Modus durchgeführt wird. Außerdem wird bei den Logs bei Nutzung des “dedicated” Modus keine Differenzierung zwischen get und write mehr vorgenommen, weshalb nur noch ein

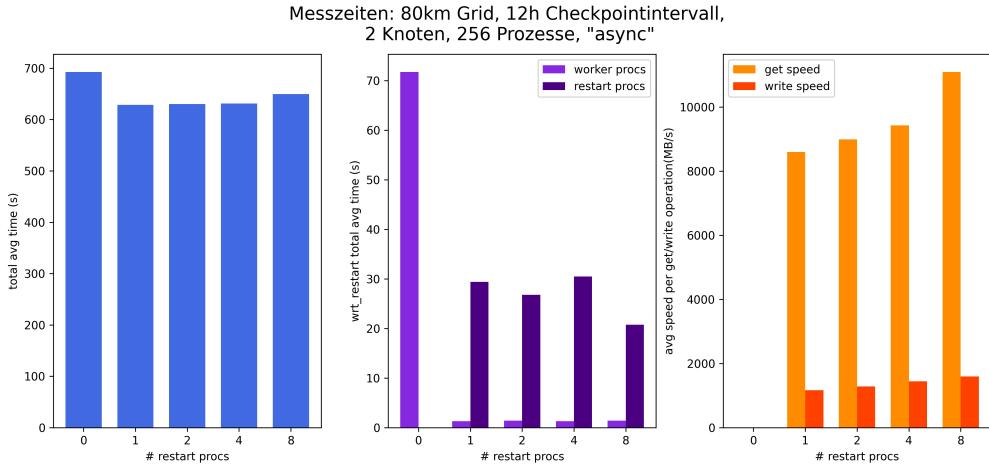
Balken in den jeweils rechten Graphen abgebildet wird. Die Messungen wurden - abgesehen von den “joint” Messungen - mit 0, 1, 2, 4, 8 Restart Procs durchgeführt.



Zunächst zu den Runs auf dem 80 km Grid. Bei einem Checkpointintervall von 12 Stunden ist deutlich an den write-Zeiten zu sehen, dass das Schreiben der Checkpoints zu infrequent für eine tatsächliche Be- oder Überlastung ist. Tatsächlich wurden bei Darshan auch nur 17,23 GiB geschriebener Output registriert, kumuliert in den von ICON generierten Logs ~16,7GiB. Folglich nimmt die absolute Rechenzeit bei den Messungen mit nur einem Knoten auch eher zu bei steigender Anzahl an Restart Procs. Erst bei 2 Knoten kann man eine leichte Verbesserung der Laufzeit sehen.

Hier wird schon eine der ersten zentralen Beobachtungen sichtbar, die besonders hervorgehoben werden sollen: das Striping, bzw. das Partitionieren des Outputs gilt nicht für das Schreiben der Checkpoints. Im “async” Modus (welcher selbst im Tutorial von 2019 [5] als “alt” bezeichnet wird) schreibt immer nur ein einzelner Prozess asynchron zu den workern. Das gute dabei ist, dass die worker weiterrechnen können, ohne auf das Schreiben der Checkpoints zu warten - gleichzeitig ist die Performance dadurch stark beschränkt. Deutlicher wird dies bei den Messungen zu dem 12-minütigen Checkpointintervall.





Zieht man die nebenstehende Heatmap aus dem Darshan Log zu Rate, betont die durchgezogene rote Linie das serielle Schreiben eines einzelnen Prozesses, obwohl eigentlich 4 Prozesse als Restart Procs dienen sollten. Insgesamt wurden in der 12 min Variante, gemessen in Darshan, knapp 850 GiB Daten geschrieben, also schon wesentlich mehr als bei den 12h Intervallen. Dies stellt sich auch in den nachstehenden Graphen dar. Im Modus "async" kann schon mit einem zusätzlichen Restart Prozess eine Verbesserung der Laufzeit erreicht werden, auch im "dedicated" Modus ist mehr als eine Halbierung der Laufzeit möglich.

Dabei wird bei letzterem auch mit Erhöhung der Zahl der Restart Procs keine wirklich signifikante weitere Optimierung mehr umgesetzt. Bei Nutzung des "joint" Modus wird in dieser Konfiguration eine ähnliche Zeit erreicht wie bei den "async" und "dedicated" Modi. Die Erhöhung der Rechenknoten auf 2 bedingt zunächst einen Anstieg der Gesamtrechenzeit bei synchronem IO im Vergleich zu dem sychronen Schreiben mit nur einem Knoten. Selbiges gilt für den Run mit 4 Knoten. Im "async" Modus ist die Laufzeit insgesamt begrenzt - es gibt keine Möglichkeit, diese weiter unter die etwa 1700s zu bekommen. Beide anderen Restart Modi weisen eine deutlich bessere Performance auf, wobei jeweils die Durchläufe mit vielen Restart Procs im "dedicated" Modus noch eine leichte Verbesserung gegenüber den jeweiligen Läufen im "joint" Modus aufweisen.

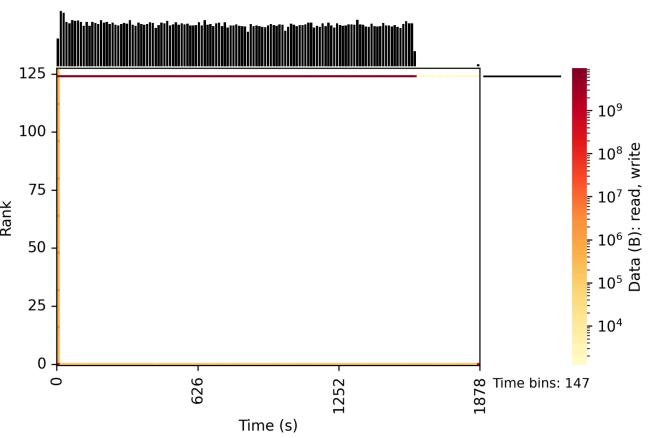
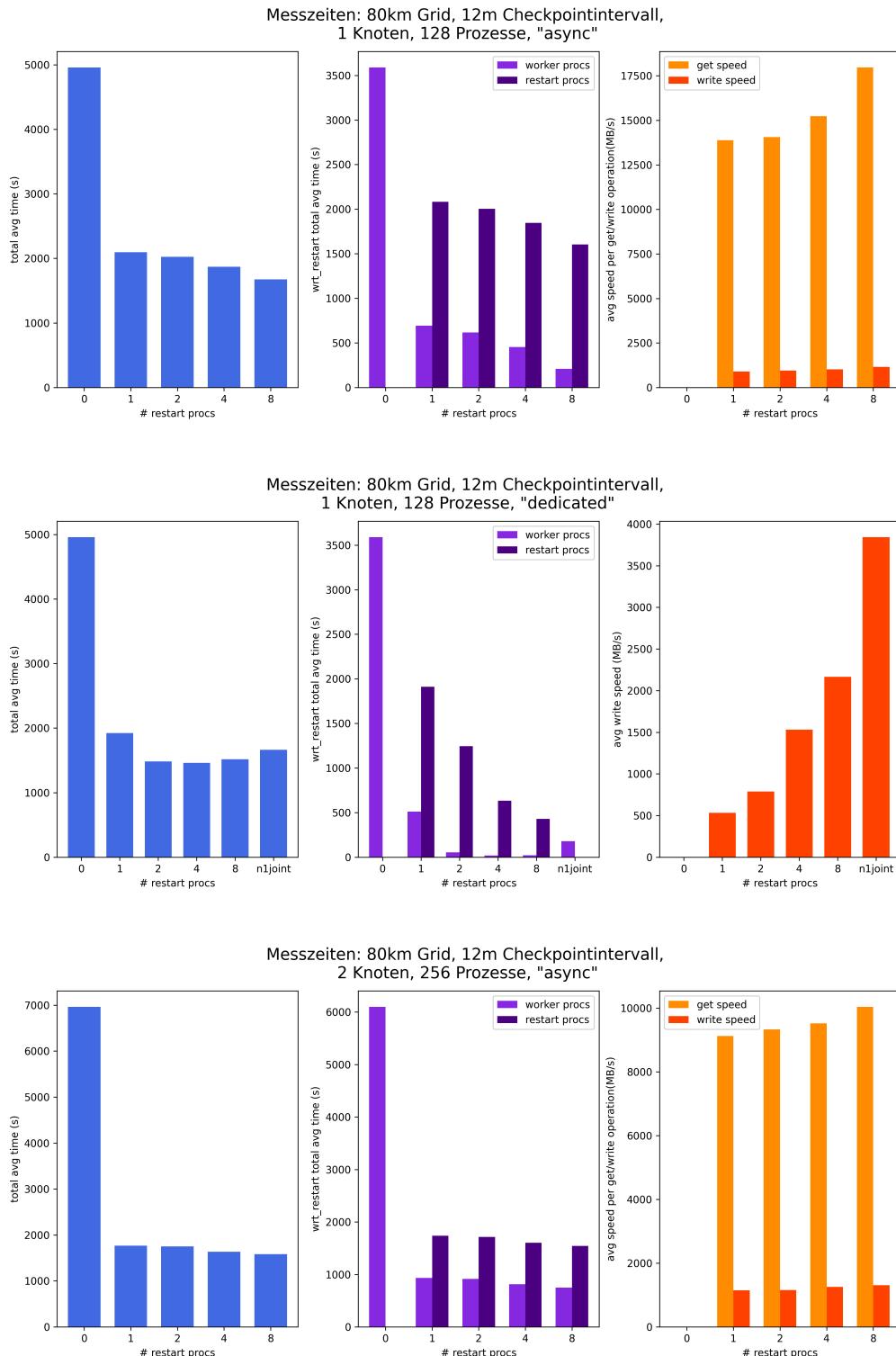
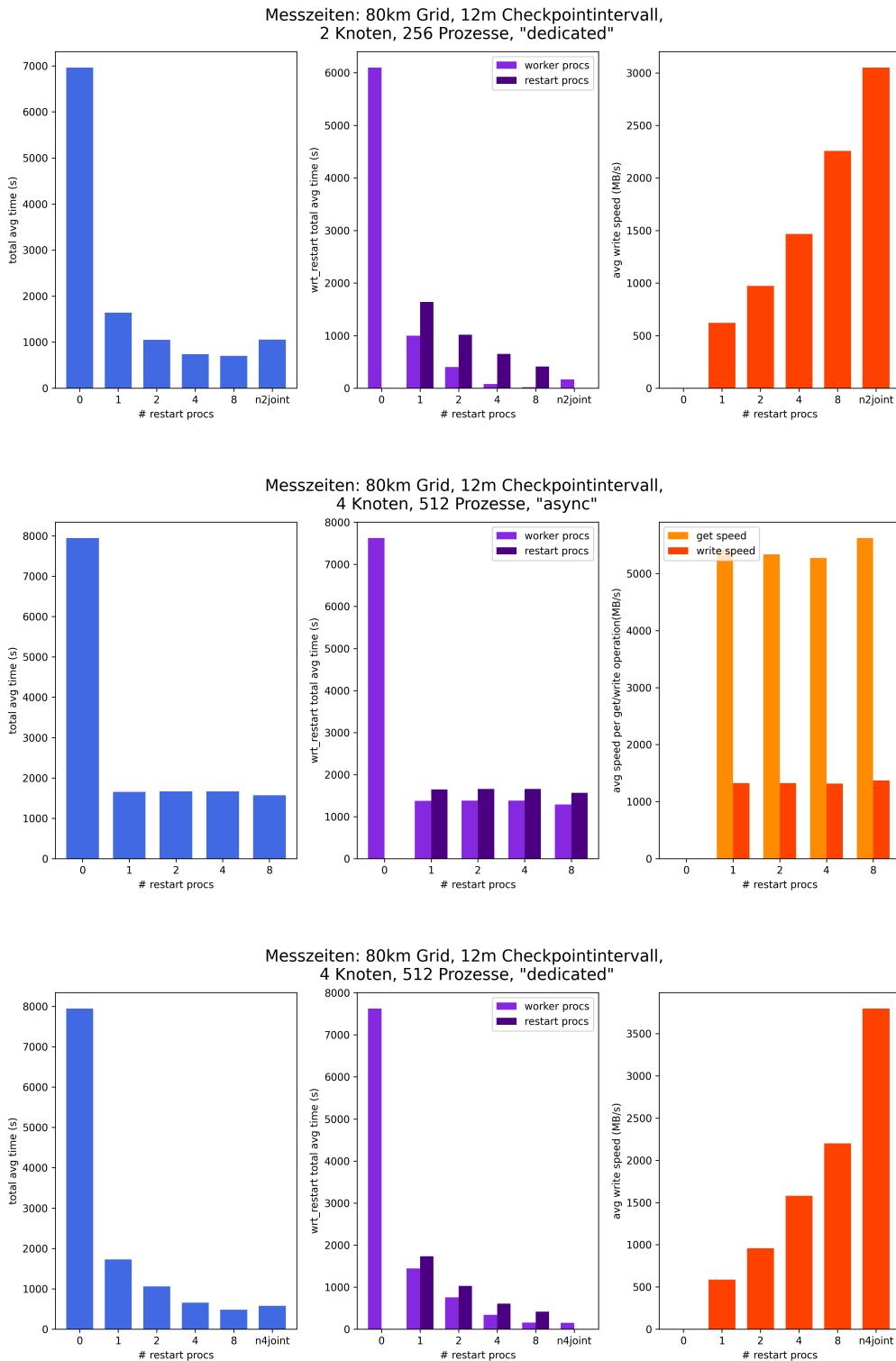
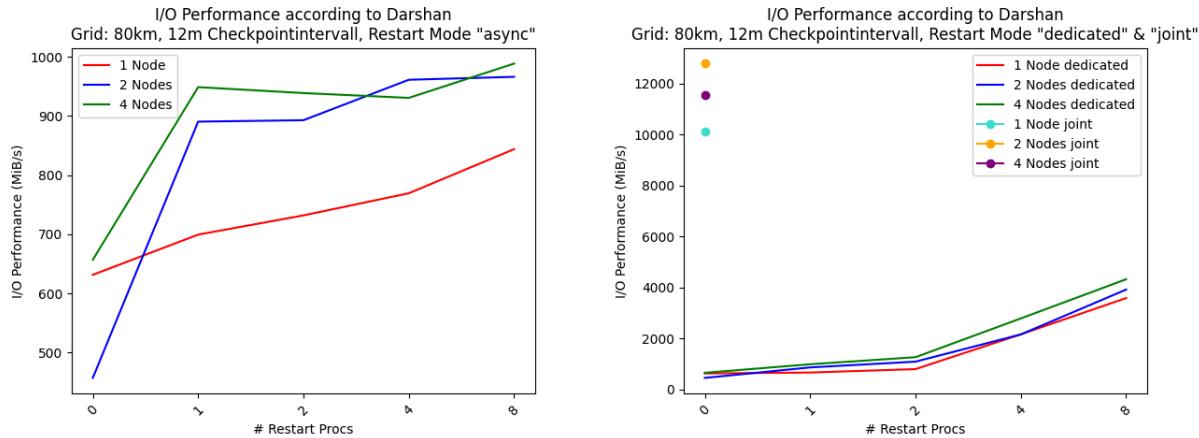


Abb. 21: Heatmap Checkpoint-IO 80km Grid, 12h Intervall, 1 Knoten, 4 Restart Procs







Die IO Performance, welche Darshan misst, ist bei den “async” Messungen einigermaßen gleichbleibend, sobald die Anzahl der Restart Procs auf mindestens 1 erhöht wird. Auffällig ist diese Metrik jedoch bei den beiden anderen Modi. Während im “dedicated” Modus erwartungsgemäß, da sich die Anzahl der Knoten, jedoch nicht der Restart Procs, erhöht, die Performance etwa gleichbleibend verläuft, tut sie dies trotz vermuteter Steigung im “joint” Modus ebenfalls (zumindest nicht mit der Verdopplung wie die Anzahl der Knoten) und liegt sowohl bei 1, 2 als auch 4 Knoten zwischen 10000 und etwa 13000 MiB/s. Die entsprechenden Heatmaps zeigen gut die Verteilung der IO-Last auf alle Prozesse (“joint”) oder nur auf die dezierten Restart Prozesse (“dedicated”).

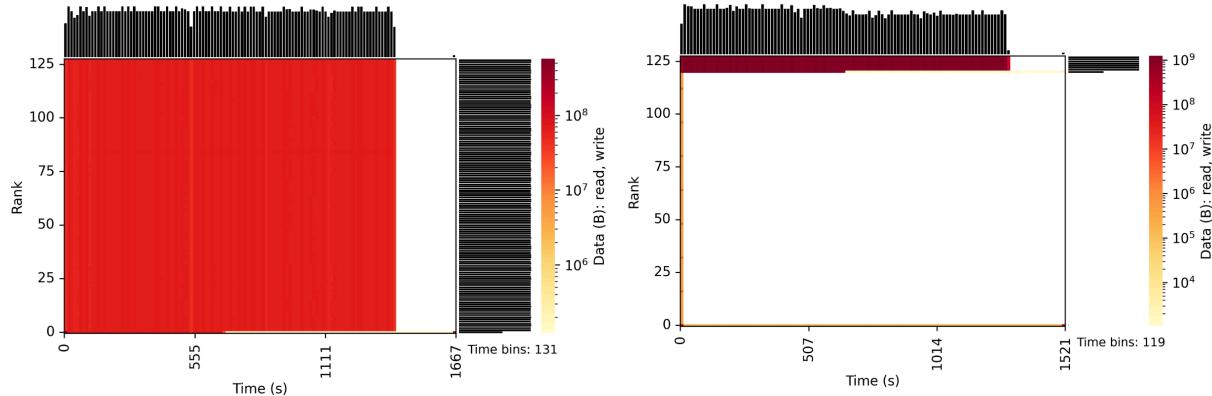


Abb. 29: Heatmaps Checkpoint-IO 80km Grid, 12m Intervall, 1 Knoten, Modi “joint” (links) und “dedicated” (rechts)

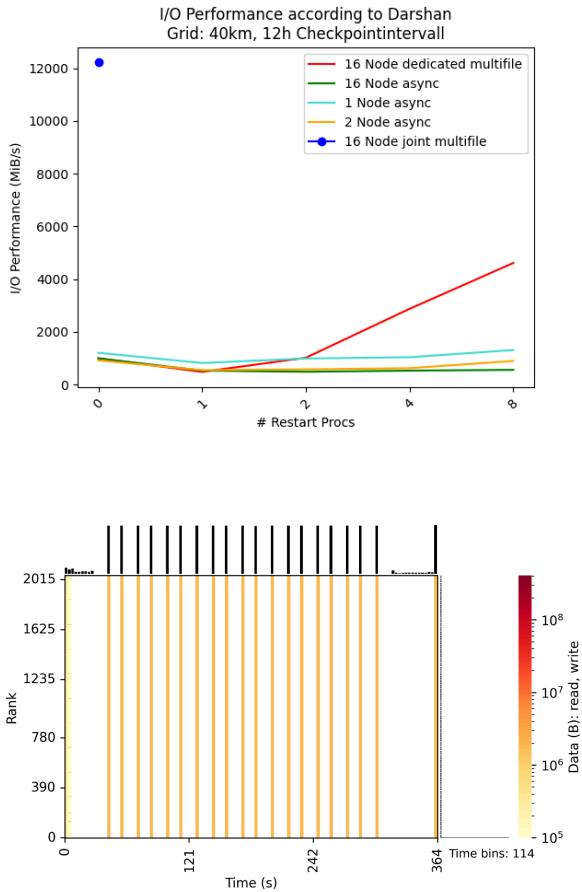
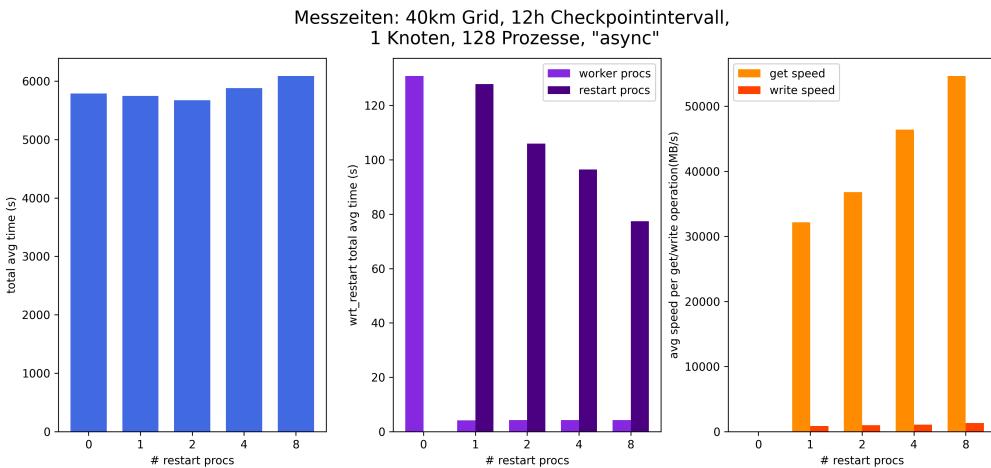


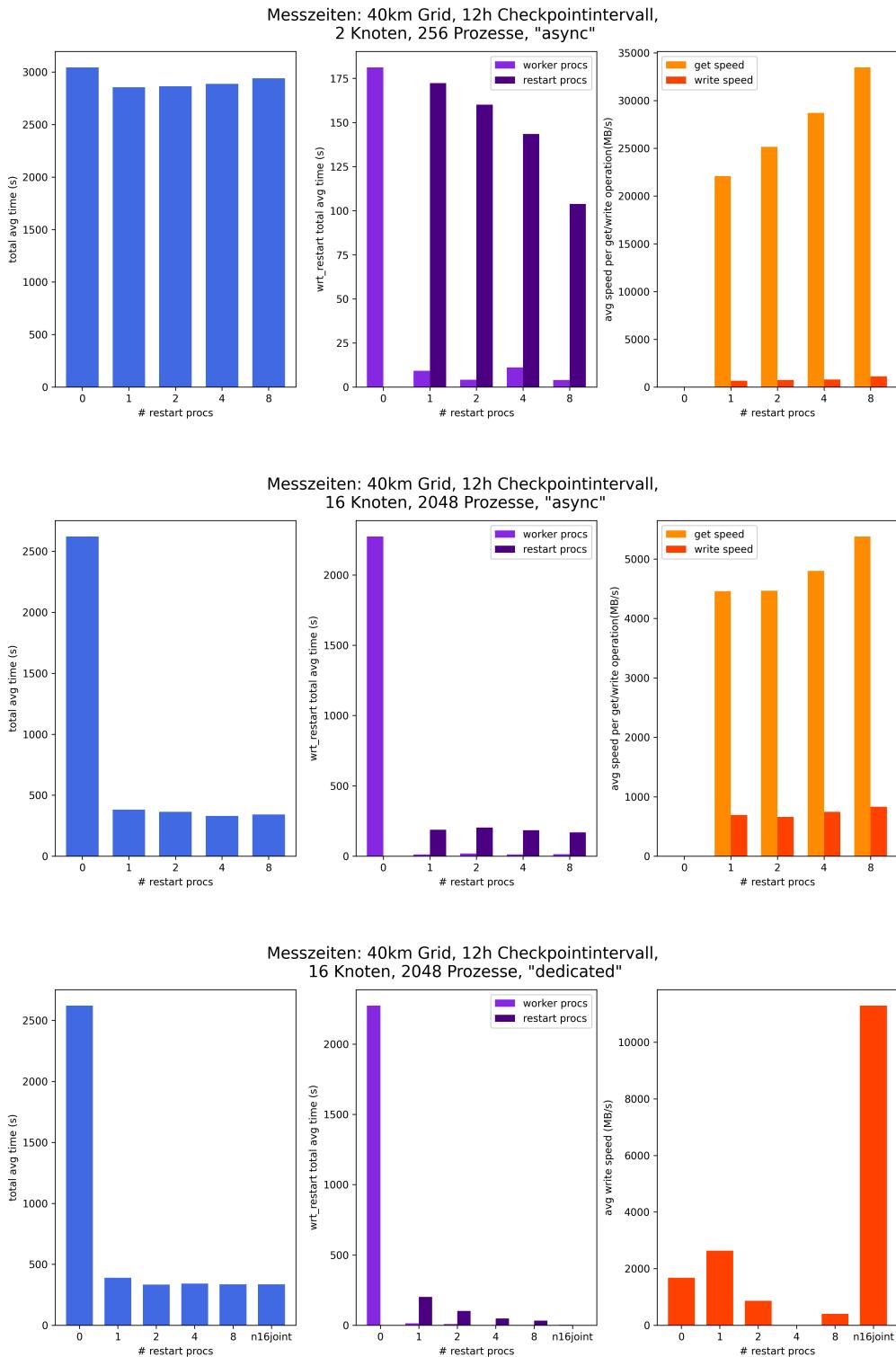
Abb. 31: Heatmap Checkpoint-IO, 40km Grid, 12h Intervall, 16 Knoten, Modus “joint”

Die Größe der geschriebenen Files beträgt je nach genauer Konfiguration um die 70 GiB. Man beachte die Heatmap (Abb. 31), in welcher genau zu sehen ist, wie alle worker Prozesse im “joint” Modus zu bestimmten Zeiten die Checkpoints schreiben.



Die Messungen zu dem 40km Grid sind mit 1 und 2 Knoten ähnlich geartet wie die auf dem 80km Grid mit 12h Intervall, da auch hier wieder eine sehr geringe Anzahl an Checkpoints geschrieben wird, und das Schreiben im Vergleich zum eigentlichen Rechnen kaum ins Gewicht fällt. Konträr dazu verhält es sich bei den Messungen mit 16 Knoten, bei denen die wrt\_restart Zeit - bei 0 Restart Procs - fast die gesamte Laufzeit in Anspruch nimmt, und im asynchronen IO nur noch etwa ein Zehntel dessen beträgt.

Die get-Geschwindigkeit ist bei den Restarts wesentlich höher als bei den eigentlichen Outputs, wobei sie mit zunehmender Anzahl an Knoten abnimmt. Die IO Performance laut Darshan bleibt bei dem 16 Knoten “joint” Durchlauf auch bei knapp über 12 MiB/s, vergleichbar mit den Runs auf dem 80km Grid. Von den anderen Runs auf dem 40km Grid ist nur im “dedicated” Modus eine Verbesserung der Performance im Zuge der Erhöhung der Restart Procs beobachtbar.



Ein paar Worte zu den Files, welche geschrieben werden, da diese Implikationen für die Nutzenden haben kann. Im "dedicated" Modus sind die eigentlichen Checkpoints als Ordner angelegt, in denen das Grid auf eine Anzahl an Files unterteilt ist, nämlich so viele wie Restart Prozesse. Gibt es 8 Restart Procs, gibt es auch 8 Files in jedem Checkpoint-Ordner, und jeder Prozess übernimmt einen eigenen Teil des Grids. Im "joint" Modus hingegen wird das Grid in so viele Teile unterteilt wie es worker Prozesse gibt, und jeder Prozess schreibt seinen Teil in den gemeinsamen Checkpoint-Ordner. Insgesamt werden also bei dem "dedicated" Modus

weniger Files geschrieben, was für Anwendende leichter zu verarbeiten sein könnte. Die Frage ist an dem Punkt, wie einfach oder schwierig es ist, die Files wieder zusammenzuführen, um am Ende das komplette Grid wiederherzustellen. Da die Files aber ohnehin nur für den internen Restart genutzt werden und nicht für sich selbst zur weiteren Bearbeitung gedacht sind, ist dies vermutlich für diese Arbeit nicht von großem Interesse.

Zuletzt sind die Aufzeichnungen aus ClusterCockpit zu betrachten. Wie bei allen CC Logs scheinen die einzelnen Knoten nur auf etwa 32 GB Memory zu laufen, trotz der eigentlich pro Knoten 256 GB zu Verfügung stehenden. Das Setup braucht um die 2 Minuten, bevor die worker anfangen, tatsächlich zu arbeiten. Dies fällt in allen Konfigurationen ähnlich aus. Fokussiert seien hier zwei Konfigurationen:

- 80km Grid, 12min Intervall, 4 Knoten, 8 Restart Procs
  - “async”: in diesem Modus leisten alle Knoten nur etwa 40 GFLOP/s. Es ist deutlich zu erkennen, dass die Lustre BW nur durch den Knoten mit den Restart Prozessen genutzt wird (bis 0,5 GB/s), und auch die IB BW hauptsächlich durch denselben Knoten ausgelastet wird (~1,75 GB/s), alle anderen Knoten sind auf Lustre bei 0 GB/s und auf IB bei ~ 0,5 GB/s. Spannenderweise ist auch die Auslastung der Memory Bandbreite bei nur etwa 30 GB/s.
  - “dedicated”: durchschnittlich etwa 135 GFLOP/s, also knapp das dreifache der Leistung des “async” Modus. Auch hier nutzt der Knoten mit den Restart Procs die Lustre BW, allerdings bis 1,4 GB/s. Die Auslastung auf InfiniBand läuft mit bis zu 6 GB/s (Knoten mit Restart Proc) bzw. bis zu 2 GB/s (Knoten ohne Restart Procs). Memory BW liegt bei etwa 100 GB/s.
  - “joint”: GFLOP/s durchschnittlich bei um die 120 GFLOP/s. Alle vier Knoten tragen gleichmäßig zur Auslastung der Bandbreite auf Lustre bei, nämlich um die 0,3 GB/s pro Knoten, kumuliert also vergleichbar mit dem “dedicated” Modus. Auffällig ist die kumuliert geringere IB BW, die sich für alle Knoten um etwa 1,25 GB/s bewegt. Dies ist Resultat der wegfallenden Datenübertragung zwischen den Knoten für das Schreiben der Files durch eigens dafür zuständige Prozesse.

Es sei daran zu erinnern, dass bei dieser Konfiguration der “dedicated” Modus, was die Rechenzeit angeht, dem “joint” Modus überlegen war, was sich auch zum Teil in den von ClusterCockpit registrierten Daten widerspiegelt.

- 40km Grid, 12h Intervall, 16 Knoten, 8 Restart Procs: durch den geringen Impact der selten zu schreibenden Checkpoints sind hier alle Metriken sehr ähnlich ausgestaltet. Es gibt keine großen Unterschiede.

Zu Testzwecken wurden die letzten Messungen mit 16 Knoten in den beiden moderneren Restart Modi noch einmal wiederholt, nur mit einem Checkpointintervall von 1h statt 12h. Trotz dieser wesentlich engmaschigeren Einstellung wurde die Laufzeit wesentlich weniger beeinträchtigt als zunächst angenommen. Von knapp über 6 Minuten Runtime erhöhte sie sich bei 1h Intervall auf um die 9 Minuten im “joint” Modus und im “dedicated” Modus mit mindestens 4 Restart Procs.

## 6 Interpretation und Diskussion

Nach all den beschriebenen Beobachtungen lässt sich feststellen, dass es verschiedene Parameter gibt, welche vor allem die Laufzeit beeinträchtigen, die hier als primär zu betrachtende

Größe im Zentrum steht. Der Einfachheit halber sollen auch an dieser Stelle die Überlegungen noch einmal in zwei Hauptkategorien unterteilt werden: Output-IO und Restart-IO.

## 6.1 Restart/Checkpoint-IO

Die durchgeführten Tests zeigen eindeutig die Überlegenheit der beiden moderneren Restart-Modi “joint procs multifile” und “dedicated procs multifile” gegenüber dem veralteten “async” Modus auf. Tatsächlich lässt sich mit dem “dedicated” Modus die Laufzeit der Experimente noch effizienter tunen als mit dem “joint” Modus, jedoch muss vorher festgelegt werden, wie viele dezidierte Prozesse festgelegt werden, um Checkpoints zu schreiben, bzw. hinterher auch Restart Files wieder einzulesen. Letzteres war nicht Teil dieser Arbeit und müsste somit noch einmal separat untersucht werden. Wenn es nur um das Schreiben geht, ist zu berücksichtigen, wie groß die einzelnen Checkpoints sind und wie oft diese geschrieben werden müssen. Falls die Rechenzeit der worker zwischen zwei Checkpoints geringer ausfällt, als die Zeit, welche die Restart Prozesse zum Schreiben benötigen, ist es sinnvoll, die Anzahl der Restart Prozesse zu erhöhen. Aufgrund der variierenden Daten aus ClusterCockpit, Darshan und den Logs, lässt sich nicht so leicht sagen, wie viel Throughput durch einen Prozess zu erwarten ist. Wäre dies bekannt, ließe sich parallel wohl auch errechnen, wie groß der zu erwartende Output an Checkpoints für eine gegebene Konfiguration eines Experiments zu erwarten ist, und somit eine Aussage treffen, wie viele Prozesse benötigt werden, um diese Checkpoints zu schreiben, ohne eine Wartezeit der worker zu provozieren. Auch dies wäre für kommende Untersuchungen ein Punkt von Interesse. Abschließend sei betont, dass der “joint” Modus sicherlich der hier zu empfehlende sei, da in diesem die Überlegung der Restart Prozesse nicht ansteht. Durch diesen werden zwar eine Unmenge an verschiedenen Dateien geschrieben, da diese jedoch ohnehin für den User im Regelfall irrelevant sind und nur für den Restart benötigt werden, ist dieses Hindernis wohl zu vernachlässigen.

## 6.2 Output-IO

Auch, wenn die Bezeichnung “Output-IO” etwas sinnbefreit scheint, sollte an dieser Stelle klar sein, dass es dabei um den Output der Ergebnisdaten des Experiments geht, ganz klar abgetrennt von den Checkpoint-Dateien. Entscheidend ist hier der Gebrauch von IO Prozessen und die Möglichkeit der Partitionierung des Outputs. Wie in Section 5.2.1 schon ausgeführt, sollte die Anzahl der Partitionen immer der Anzahl der dezidierten IO Procs entsprechen. Hier lässt sich an die Beobachtungen aus dem letzten Kapitel anschließen: Für eine klare Aussage, wie viele dezidierte Prozesse für IO reserviert werden sollten, muss zunächst der eigentlich anfallende Throughput bekannt sein, d.h. Größe einzelner Files und Häufigkeit der Outputs. Dazu ist wichtig:

1. wie viele Knoten X werden genutzt?
2. wie lange brauchen X Knoten zur Berechnung zwischen einzelnen Zeitschritten des Experiments?
3. wie oft fallen Outputs an (alle Y Zeitschritte)?
4. wie groß sind die Outputs (Z zu schreibende Variablen)?
5. wie viel MB an Daten können in einer gegebenen Zeit an einen Prozess übertragen werden?
6. wie viel MB an Daten kann ein IO Prozess in einer gegebenen Zeit schreiben?

Punkt 1 und 3 sind einfache Einstellungen, welche im Jobskript vorgenommen werden. Punkt 2 ist im Wesentlichen abhängig von der Größe des Grids, d.h. der Anzahl der zu berechnenden

Datenpunkte. Bisher ist in dieser Arbeit nicht klar geworden, ob verschiedene Experimente, die aber alle auf dem gleichen Grid laufen, ohne Betrachtung des IO gleich viel Laufzeit benötigen - was sich leicht in weiteren Arbeitsschritten beantworten ließe. Vermutlich ließe sich diese Zeit jedoch im Voraus berechnen, soweit die getroffenen Einstellungen aus dem Experimentskript vorliegen. Punkt 4 ist eine der entscheidendsten Informationen, die an dieser Stelle benötigt werden. In Testläufen wurde versucht, durch Isolation die Größe einzelner Variablen im Output herauszufinden, leider erfolglos. Dies kommt im wesentlichen auf die Datenformate an. Welche Variablen werden als Float festgehalten, welche als Double oder Integer? Werden teilweise vielleicht auch Variablen in Arrays gespeichert? Eine Aufarbeitung dessen wird vermutlich mit dem entsprechenden Expert\*innenwissen einhergehen müssen, welches am DKRZ vorhanden ist. Punkte 5 und 6 sind analog zu den im Ergebniskapitel zu den Checkpoints festgehaltenen Überlegungen zu betrachten. Die Messungen und festgehaltenen Daten gehen stark auseinander, sodass es schwer ist, eine klare Aussage zu treffen. Dies könnte Teil einer tiefergehenden Auseinandersetzung mit dem Thema sein, die hier nicht mehr behandelt werden kann, aber in Absprache mit den Expert\*innen am DKRZ vielleicht noch einmal angegangen werden könnte, sollte Bedarf bestehen. Die Ausarbeitung dieser Fragen würde in meiner Vorstellung eine Art Tool ermöglichen, welches vor Start eines Runs aufgrund von ausgewählten Einstellungen die optimale Anzahl an IO Prozessen vorschlägt, ohne ein Bottleneck-Problem hervorzurufen.

Darüber hinaus, ist ein weiteres Feld, welches tiefergehende Betrachtung verdient, die Problematik, dass mehr Outputstreams auch mehr Files bedeuten. Bei Priorisierung der Nutzer\*innenfreundlichkeit sollte sich damit befasst werden, wie diese Files effizient zu einer Datei zusammengefügt werden können, falls dies nicht ohnehin schon geschehen ist. Als lohnenswert würde sich vermutlich auch die Auseinandersetzung mit Score-P in Verbindung mit den hier durchgeföhrten Testruns herausstellen, um näher darauf eingehen zu können, wie die Kommunikation zwischen den Knoten und diversen Prozessen ausgestaltet wird. Bislang sind sowohl Restart- als auch IO-Prozesse immer auf dem ranghöchsten Knoten zu finden - möglicherweise wäre auch die Verteilung auf die einzelnen Knoten jedoch eine gute Variante, um die IO Performance zu steigern.

Alle benutzten Skripte, alle Ressourcen, Darshan-Logs, ICON-Logs etc. sind im entsprechenden Git Repository [12] zu finden.

## 7 Herausforderungen und Limitationen

Zum Schluss ist zu sagen, dass die Einarbeitung in ICON als Bachelorstudierender ohne große Vorerfahrungen mit der Programmiersprache (Fortran mitsamt entsprechender Namelists), einer Codebasis solch umfassender Größe, der teils veralteten oder unklaren Dokumentation und der Vielzahl an Möglichkeiten, die ICON an Einstellungen bietet, eine große Herausforderung darstellte. An vielen Stellen gäbe es hier wohl Optimierungsmöglichkeiten, obwohl klar ist, dass ICON primär der Arbeit von Expert\*innen der Klimadomäne dient und eher pragmatisch zu betrachten ist. Bei tiefergehender Arbeit im eigentlichen Code in Fortran würde dies wahrscheinlich noch offensichtlicher werden. Viele Erkenntnisse mussten erst mit längerem Trial and Error Vorgehen getroffen werden, welche anderweitig wohl schon bekannt gewesen wären, aber nicht Teil der Dokumentation waren.

Dies wird auch an der Stelle des Auslesens der ICON-Logs zur Extraktion der Zeitdaten deutlich. Jeder Restart Modus hat eine eigene, von den anderen Modi abweichende Art, diese Daten zu formattieren, sodass nicht einfach ein Skript alles auslesen kann, sondern dieses immer entsprechend umgeschrieben werden muss. Außerdem weichen die Daten zwischen ClusterCockpit, Darshan und den von ICON ausgegebenen Logs voneinander ab. Vermutlich sind diese mit mehr Hintergrundwissen zu vereinen, nach der umfassenden bisherigen Einarbeitung allerdings trotzdem noch eine weitere Schwierigkeit.

## 8 Versicherung an Eides Statt

Name: Florian Ott

Die vorliegende Arbeit wurde unter meiner Versicherung mit keiner außer den in der Arbeit und den dazugehörigen Materialien angegeben Hilfsmitteln und Quellen verfasst. Dazu zählen auch das unter [13] genannte Template für Typst, einer neuen Alternative zu LaTeX, und Copilot, einem KI-gestützten Schreibassistenten, welcher bei groben ersten Konstrukten für die Skripte hilfreich war, bevor diese für meinen individuellen Bedarf ausgearbeitet wurden.

Weiter erkläre ich, dass diese Arbeit bislang noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Hamburg, 31.03.2025  


## 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](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\\_\(Wettervorhersagemodell\)](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](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](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](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](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](https://wiki.lustre.org/images/f/f9/Shared-File-Performance-in-Lustre_Farrell.pdf)
- [12] F. Ott, “Git Repository dieses Projekts.” [Online]. Available: <https://github.com/ottfl/projekt>
- [13] “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>