

Seminar Report

Streaming Data Analytics vs Batch-Processing in HPC Systemen

Annika Marie Günther

MatrNr: 22627729

Supervisor: Sepehr Mahmoodian

Georg-August-Universität Göttingen
Institute of Computer Science

17. August 2026

Abstract

High-Performance Computing (HPC) Systeme werden zunehmend mit der Verarbeitung massiver Datensätze konfrontiert, die durch wissenschaftliche Simulationen erzeugt werden. Ein kritischer Engpass in diesen Workflows ist häufig nicht die Berechnung selbst, sondern die Input/Output (I/O)-Operationen, die erforderlich sind, um Zwischen- und Endergebnisse zu speichern und abzurufen. Dieser Seminarbericht untersucht die Leistungsauswirkungen zweier unterschiedlicher Datenverarbeitungsparadigmen in HPC-Umgebungen: Batch-Processing und Stream Processing. Während Batch-Processing aufgrund seiner Einfachheit und Kompatibilität mit Job-Schedulern für viele wissenschaftliche Workflows der Standard bleibt, bietet Stream Processing das Potenzial für reduzierte Latenz und In-situ-Analysen, wodurch I/O-Engpässe verringert werden können.

Unter Verwendung einer zweidimensionalen Wärmediffusionssimulation als repräsentativen Anwendungsfall implementiert und vergleicht diese Arbeit beide Ansätze unter Verwendung des Adaptable Input/Output System version 2 (ADIOS2)-Frameworks innerhalb der JupyterHPC-Umgebung. Die Bewertung konzentriert sich auf drei Schlüsselmetriken: Latenz, Gesamtlaufzeit und Skalierbarkeit. Die Ergebnisse zeigen, dass Stream Processing die Latenz im Vergleich zu Batch-Processing um bis zu 98 % reduzieren kann, was nahezu Echtzeit-Einblicke ermöglicht. Darüber hinaus erreicht Stream Processing bei großen Datenmengen eine 43 %ige Verkürzung der gesamten Workflow-Laufzeit durch die Überlappung von Berechnung und Analyse. Die Arbeit identifiziert jedoch auch Overhead-Kosten, die mit Streaming-Pipelines verbunden sind, was Batch-Processing für kleine Datensätze oder Szenarien, in denen keine Echtzeit-Fähigkeit erforderlich ist, effizienter macht. Dieser Bericht schließt mit Empfehlungen für die Auswahl des geeigneten Verarbeitungsmodells basierend auf Datenvolumen und Latenzanforderungen in wissenschaftlichen HPC-Workflows ab.

Transparenzerklärung zur Nutzung von KI

- **Wie wurde KI in der Ideenfindung verwendet?** *Antwort: KI-Tools wurden verwendet, um potenzielle Anwendungsfälle und Struktur zu brainstormen*
- **Wie wurde KI bei der Arbeit mit Literatur und Quellen verwendet?** *Antwort: KI-Tools halfen beim Verständnis (insbesondere dem Übersetzen) von Literatur*
- **Wie wurde KI im Prozess des akademischen Schreibens verwendet?** *Antwort: KI-Tools wurden zum Korrekturlesen und zur Optimierung verwendet*
- **Wie wurde KI in der Methodik, insbesondere bei der Datenerhebung und -analyse, eingesetzt?** *Antwort: KI-Tools wurden zur Unterstützung bei der Codeentwicklung verwendet*
- **Wie wurde KI sonst verwendet?** *Antwort: -*

Ich erkläre hiermit, dass alle Nutzungen von KI-Tools in dieser Arbeit vollständig und genau offengelegt wurden. Fehlende oder falsche Informationen können als Täuschungsversuch gewertet werden.

Inhaltsverzeichnis

Tabellenverzeichnis	iv
Abbildungsverzeichnis	v
Abkürzungsverzeichnis	vi
1 Einleitung	1
2 Hintergrund	2
2.1 Was ist Batch-Processing	2
2.2 Was ist Stream Processing	2
2.3 Batch vs. Streaming	3
2.4 HPC und Batch/Stream Processing	3
3 Werkzeug- und Plattformauswahl	4
3.1 Vergleich von Streaming-Frameworks	4
3.2 Plattformauswahl	5
4 Methodik und Systemarchitektur	6
4.1 Das Simulationsmodell	6
4.2 Datenfluss und ADIOS2 Integration	7
4.3 Evaluierungsmetriken und Messmethodik	8
5 Ergebnisse	9
5.1 Latenzanalyse	9
5.2 Laufzeitanalyse	10
5.3 Skalierbarkeit und Komponentenanalyse	11
6 Diskussion	12
6.1 Interpretation der Leistungsvorteile	13
6.2 Praktische Implikationen für HPC-Workflows	13
6.3 Grenzen der Studie	14
7 Fazit	14
Literatur	16
A Batch Processing Producer Code	A1
B Stream Processing Producer Code	A2
C Batch Processing Consumer Code	A3
D Stream Processing Consumer Code	A4

Tabellenverzeichnis

1	Vergleich Batch vs. Stream Processing	3
2	Vergleich der Streaming-Frameworks	5
3	Vergleich der Latenz (in Sekunden)	9
4	Vergleich der Gesamtlaufzeit (in Sekunden)	10
5	Komponentenanalyse der Laufzeiten (in Sekunden)	11
6	Zusammenfassung der Handlungsempfehlungen	15

Abbildungsverzeichnis

1	Workflow-Vergleich Batch vs. Streaming	8
2	Latenzvergleich	10
3	Gesamtlaufzeitvergleich	11
4	Komponentenanalyse	12

Abkürzungsverzeichnis

ADIOS2 Adaptable Input/Output System version 2

API Application Programming Interface

BP Buffer Protocol

HDF5 Hierarchical Data Format 5

HPC High-Performance Computing

IoT Internet of Things

I/O Input/Output

MPI Message Passing Interface

SD Standardabweichung

1 Einleitung

HPC-Systeme werden eingesetzt, um rechenintensive wissenschaftliche Simulationen auszuführen, wobei neben der Berechnung selbst auch die Verarbeitung der entstehenden Daten eine zentrale Rolle spielt [Ali+25]. Die Leistungsfähigkeit moderner Supercomputer wird zunehmend durch die Geschwindigkeit der Datenbewegung limitiert, nicht mehr primär durch die Rechengeschwindigkeit der Prozessoren. Gerade bei datenintensiven Simulationen kann die Notwendigkeit, große Zwischen- und Endergebnisse zunächst auf das Dateisystem zu schreiben, zu I/O-Engpässen führen und damit die Gesamtleistung eines Workflows begrenzen [Ihd+22]. Dieses Phänomen wird in der Fachliteratur oft als "I/O-Wall" bezeichnet. Vor diesem Hintergrund gewinnt die Frage an Bedeutung, ob Simulationsdaten erst nach Abschluss eines Laufs gesammelt analysiert werden sollten oder ob eine Verarbeitung bereits während der Erzeugung sinnvoller ist.

Das zentrale Problem dieser Arbeit besteht im Vergleich zweier Verarbeitungsmodelle für HPC-Workflows: Batch-Processing und Stream Processing. Batch-Processing eignet sich gut für abgeschlossene Datenmengen und ist in HPC-Umgebungen etabliert, kann aber eine höhere Latenz verursachen, weil Analyse und Berechnung zeitlich voneinander getrennt sind [Vin24]. Stream Processing setzt dagegen auf die kontinuierliche Verarbeitung eingehender Daten und kann dadurch frühere Auswertungen ermöglichen, bringt jedoch zusätzliche Anforderungen an Koordination, Zustandsverwaltung und Systemintegration mit sich [LCJ18]. Die Relevanz dieses Vergleichs liegt darin, dass moderne HPC-Workflows nicht nur Rechenleistung benötigen, sondern auch effiziente Datenbewegung und flexible Analysepfade [Ali+25].

Als etablierte Lösung in wissenschaftlichen Workflows wird häufig weiterhin ein batchorientierter Ansatz verwendet, weil er sich gut mit klassischen Job-Schedulern und reproduzierbaren Rechenläufen kombinieren lässt [Ali+25]. Für Streaming- und In-situ-Szenarien stehen dagegen spezialisierte Werkzeuge wie ADIOS2 zur Verfügung, das ausdrücklich für skalierbares I/O und datenstromorientierte wissenschaftliche Anwendungen entwickelt wurde [ADIOS]. Allgemeine Messaging- oder Streaming-Plattformen können zwar technisch ähnliche Funktionen bieten, sind aber für HPC-Workflows oft zu allgemein gehalten oder erfordern zusätzliche Infrastruktur, die für einen fokussierten Vergleich nicht notwendig ist [flink; NATS25; Pulsar]. Die Literatur zeigt daher, dass der Übergang von dateibasierten Workflows zu Streaming-Pipelines vor allem dann interessant ist, wenn Dateisystemgrenzen und Datenbewegung zum Engpass werden.

Die Arbeit untersucht diesen Zusammenhang anhand einer zweidimensionalen Wärmediffusionssimulation als Beispiel für einen wissenschaftlichen HPC-Workflow mit kontinuierlich erzeugten Daten. Die Simulationsdaten werden dabei entweder gesammelt als Batch-Variante oder schrittweise in einer Streaming-Variante über ADIOS2 verarbeitet. Daraus ergibt sich die Forschungsfrage:

Wie unterscheidet sich streaming-basierte Verarbeitung von batch-basierter Verarbeitung für Simulationsdaten in HPC-Workflows hinsichtlich Latenz, Gesamtlaufzeit und Skalierbarkeit?

Ziel ist es, zu prüfen, ob Streaming im gewählten Anwendungsfall messbare Vorteile bei Latenz, Gesamtlaufzeit und Skalierbarkeit bietet und welche praktischen Grenzen sich daraus ergeben. Die Untersuchung soll damit zeigen, in welchen Fällen Streaming gegenüber Batch-Processing Vorteile bietet und in welchen Fällen der klassische batchorientierte Ansatz weiterhin angemessen ist.

Beiträge der Arbeit

- Entwicklung eines reproduzierbaren, HPC-konformen Vergleichsmodell für Batch- und Stream-Verarbeitung unter Verwendung von ADIOS2 und JupyterHPC
- Implementierung einer realistischen Anwendungssimulation mit klar definierten Metriken zur Leistungsbewertung
- Systematischer Vergleich der beiden Verarbeitungsmodelle anhand von 3 Metriken (Latenz, Gesamtlaufzeit, Skalierbarkeit)

Der restliche Bericht ist wie folgt gegliedert. Kapitel 2 führt die Grundlagen zu Batch-Processing, Stream Processing und deren Bedeutung im HPC-Kontext ein. Kapitel 3 vergleicht relevante Werkzeuge und begründet die Wahl von ADIOS2 und JupyterHPC. Kapitel 4 beschreibt die Methodik und Systemarchitektur, einschließlich des Simulationsmodells, der Datenfluss-Integration sowie der Evaluierungsmetriken und Hypothesen. Kapitel 5 präsentiert die quantitativen Ergebnisse der Experimente. Kapitel 6 diskutiert diese Ergebnisse im Kontext verwandter Arbeiten und praktischer Implikationen. Kapitel 7 schließt die Arbeit mit einem Fazit und einem Ausblick ab.

2 Hintergrund

2.1 Was ist Batch-Processing

Batch-Processing bezeichnet die Verarbeitung von Daten in abgeschlossenen, endlichen Einheiten, die vollständig gesammelt und anschließend in Blöcken verarbeitet werden [Vin24]. Dieses Verfahren eignet sich besonders für Szenarien, in denen die Datenmenge im Voraus bekannt ist und keine Echtzeitverarbeitung erforderlich ist. Große Datenmengen werden in Batches abgearbeitet, was bei datenintensiven Anwendungen wie etwa Offline-Analysen, periodische Auswertungen oder wissenschaftlichen Workflows effizient ist [Bel+20]. Nach der Konfiguration läuft Batch-Processing weitgehend automatisiert, wobei der manuelle Aufwand auf die Job-Planung beschränkt bleibt. Der Ansatz ist jedoch unflexibel: Nach Job-Start sind Änderungen meist nicht möglich. Batch-Processing eignet sich daher für *bounded data* mit klarem Anfang und Ende [Vin24]. In der Praxis bedeutet dies, dass ein Job gestartet wird, alle Daten verarbeitet werden und erst am Ende ein Ergebnis vorliegt.

2.2 Was ist Stream Processing

Stream Processing wurde für Echtzeit-Datenanalysen entwickelt. Daten werden kontinuierlich und schrittweise verarbeitet, während sie durch das System fließen, das heißt sie werden direkt nach Empfang bearbeitet [LCJ18; Eis+24]. Dieser Ansatz ist besonders nützlich, wenn die Datengröße unbekannt oder effektiv unbegrenzt ist, etwa bei Daten aus Sensoren oder anderen Live-Quellen [LCJ18; Poe+22]. Stream Processing arbeitet mit *unbounded data*, also Datenströmen ohne festes Ende [LCJ18; Vin24]. Dies erfordert Infrastruktur für Zustandsverwaltung, Fehlertoleranz und variable Datenraten [LCJ18]. In wissenschaftlichen Anwendungen wird Stream Processing relevant, wenn hohe Datenraten während der Laufzeit ausgewertet werden müssen [Poe+22]. Ein zentrales Konzept hierbei

ist die In-situ-Analyse, bei der die Analyse direkt im Speicher der Simulation stattfindet, ohne dass Daten auf ein Dateisystem geschrieben werden müssen.

2.3 Batch vs. Streaming

Der wichtigste Unterschied zwischen beiden Ansätzen ist die Latenz. Batch-Processing weist eine höhere Latenz auf [LCJ18], während Stream Processing eine deutlich niedrigere Latenz bietet [Eis+24]. Batch-Processing ist generell einfacher und weniger komplex zu implementieren [Vin24], während Streaming-Systeme komplexer sind, da sie kontinuierlichen Input, Zustandsverwaltung, Timing und Fehlertoleranz bewältigen müssen [LCJ18]. Batch eignet sich hervorragend für große Offline-Workloads und periodische Berichterstattung [Bel+20], während Streaming besser für Echtzeit-Einblicke und unmittelbare Reaktionen geeignet ist [Poe+22]. Hinsichtlich der Infrastruktur und Kosten sind Batch-Systeme weniger anspruchsvoll [Vin24], während Streaming-Systeme eine spezialisierte Infrastruktur und mehr Rechenressourcen benötigen [LCJ18].

Tabelle 1: Vergleich Batch vs. Stream Processing

Kriterium	Batch Processing	Stream Processing
Latenz	Hoch	Niedrig
Datenvolumen	Große, endliche Mengen	Kontinuierliche Ströme
Komplexität	Einfach	Hoch
Use Cases	Offline-Analyse	Echtzeit-Monitoring
Infrastruktur	Standard	Spezialisiert

2.4 HPC und Batch/Stream Processing

HPC bezeichnet die Nutzung paralleler Rechnersysteme wie Cluster oder Supercomputer zur Lösung rechenintensiver Probleme [Ali+25]. HPC-Systeme zeichnen sich durch hohe Rechenleistung, große Arbeitsspeicherkapazitäten und effiziente Kommunikation zwischen Rechenknoten aus [Ihd+22]. Sie finden vor allem in wissenschaftlichen Simulationen, datenintensiven Analysen und komplexer Modellierung Anwendung [Ali+25]. HPC-Anwendungen erzeugen große Datenmengen als Zwischen- und Endergebnisse [Ihd+22], deren Verarbeitung Leistung und Skalierbarkeit direkt beeinflusst [Poe+22].

Für HPC-Systeme sind die Konzepte von Batch und Streaming von besonderer Bedeutung, da HPC-Anwendungen häufig sehr große Datenmengen erzeugen und verarbeiten. Klassische HPC-Workflows sind meist batch-orientiert, da Simulationen oder Rechenläufe typischerweise in einer klar begrenzten Zeit ausgeführt werden und die Ergebnisse erst im Anschluss analysiert werden [Ali+25]. Dieses Vorgehen ist für viele wissenschaftliche Anwendungen geeignet, bei denen zunächst die Berechnung selbst im Vordergrund steht und die Auswertung in einem separaten Schritt erfolgt. Gleichzeitig gewinnt Stream Processing in HPC zunehmend an Relevanz, weil moderne wissenschaftliche Anwendungen Daten bereits während der Laufzeit analysieren möchten, etwa bei Instrumenten, Messsystemen oder simulationsbasierten Workflows mit hohem Datenaufkommen [LCJ18; Poe+22].

Ein zentrales Problem klassischer dateibasierter HPC-Workflows ist die Belastung des Dateisystems. Wenn große Datenmengen zunächst geschrieben und anschließend wieder gelesen werden müssen, entstehen zusätzlichen I/O-Kosten und potenzielle Engpässe [Poe+22; Fre+23]. Streaming-Ansätze können hier einen Vorteil bieten, indem sie Daten

direkt an Analysekomponenten weiterleiten und so die Abhängigkeit von einem klassischen dateibasierten Zwischenschritt reduzieren [Vin24; Poe+22]. Dies ist insbesondere dann relevant, wenn Ergebnisse frühzeitig benötigt werden oder wenn der Speicherzugriff zum begrenzenden Faktor wird [Vin24]. Pilot-Streaming [LCJ18] verdeutlicht beispielsweise, dass Streaming-Anwendungen auf HPC-Infrastruktur besondere Anforderungen an Ressourcenmanagement, Kommunikation und die Entkopplung von Verarbeitungskomponenten stellen. Gleichzeitig zeigt die Literatur, dass moderne HPC-Systeme nicht nur auf Rechenleistung, sondern zunehmend auch auf effiziente Datenverarbeitung und Datenbewegung ausgelegt sein müssen [Ihd+22; Poe+22]. Insgesamt wird deutlich, dass Batch- und Stream Processing im HPC-Kontext keine konkurrierenden, sondern komplementäre Verarbeitungsmodelle darstellen. Batch-Processing bleibt für abgeschlossene Simulationen und nachgelagerte Analysen weiterhin zentral, während Streaming insbesondere dort an Bedeutung gewinnt, wo Daten während der Berechnung ausgewertet oder frühzeitig weiterverarbeitet werden sollen. Genau deshalb ist der Vergleich beider Ansätze für wissenschaftliche HPC-Workflows von hoher Relevanz.

3 Werkzeug- und Plattformauswahl

3.1 Vergleich von Streaming-Frameworks

Für HPC-Workflows kommen mehrere Tools infrage, die ich im Folgenden vergleiche:

Apache Flink [flink] ist eine Open-Source-Streaming-Engine, die sowohl Batch- als auch Stream-Verarbeitung in einem einzigen Framework unterstützt. Sie ist konzipiert für zustandsbehaftete, ereignisgesteuerte Anwendungen mit hoher Durchsatz- und Latenz-Performance. Flink [flink] ist besonders bekannt für seine exactly-once Semantik und Windowing-Funktionen, die es ermöglichen, komplexe zeitbasierte Analysen in Echtzeit durchzuführen. Flink [flink] wird häufig in Cloud- und Unternehmensumgebungen eingesetzt, insbesondere für Anwendungen wie Echtzeit-Berichterstattung, Internet of Things (IoT)-Datenverarbeitung und Web-Log-Analysen. Trotz seiner Leistungsfähigkeit ist Flink für HPC-Workflows nicht ideal. Die Einrichtung erfordert zusätzliche Infrastruktur, und die Integration in Job-Scheduler ist komplex. Für ein Seminarprojekt mit begrenzter Zeit ist Flink zu aufwändig und überdimensioniert. Es ist nicht spezialisiert auf wissenschaftliche Daten, sondern auf allgemeine Event-Streaming-Anwendungen.

NATS JetStream [NATS25] ist ein leichtgewichtiges, hoch performantes Nachrichtensystem mit persistenter Speicherung und Replay-Fähigkeit. Es ist bekannt für seine geringe Latenz, einfache Application Programming Interface (API) und hohe Skalierbarkeit. NATS JetStream [NATS25] wird typischerweise in Microservice-Architekturen eingesetzt. Für HPC-Workflows ist NATS nicht geeignet, da die Unterstützung für große Datenstrukturen wie 2D-Arrays oder wissenschaftliche Metadaten fehlt. Es ist kein HPC-Tool, sondern ein universelles Nachrichtensystem. Die Integration in HPC-Cluster ist möglich, aber nicht optimiert.

Apache Pulsar [Pulsar] ist ein verteiltes Nachrichtensystem mit Mehrbenutzerfähigkeit und persistenter Speicherung. Es ist für Anwendungen mit hohem Datendurchsatz konzipiert und kombiniert das Herausgeber-Abonnent-Muster mit Warteschlangen-Verwaltung. Pulsar [Pulsar] wird in großen Cloud- und Unternehmensumgebungen eingesetzt. Trotz seiner Stärken ist Pulsar nicht für HPC-Workflows geeignet. Die Einrichtung erfordert zusätzliche Cluster-Infrastruktur, und die Integration in Job-Scheduler ist kom-

plex. Es ist kein HPC-optimiertes Framework, sondern ein Cloud- und Enterprise-Tool.

ADIOS2 [ADIOS] ist ein HPC-optimiertes Framework für die effiziente wissenschaftliche Datenbewegung. Es unterstützt dateibasiertes I/O sowie In-situ-Analyse und In-transit-Analysen und ermöglicht es, Simulationsdaten zwischen Producer- und Consumer-Prozessen zu schreiben, zu lesen und zu übertragen. ADIOS2 [ADIOS] ist speziell für HPC-Workflows entwickelt und integriert sich nahtlos in HPC-Jobs. Es ermöglicht Direktzugriff und Ein-/Ausgabe mit niedriger Latenz.

Die Entscheidung für ADIOS2 basiert auf drei konkreten Anforderungen an das Tool. Erstens: Das Tool muss eine nahtlose HPC-Integration bieten. Zweitens: Es darf keine zusätzliche Cluster-Infrastruktur erfordern, da es den Aufwand für ein Seminarprojekt unnötig erhöhen würde. Drittens: Es muss eine niedrige Latenz bei In-situ oder In-Transit Datenbewegungen ermöglichen. ADIOS2 erfüllt alle diese Anforderungen.

Tabelle 2: Vergleich der Streaming-Frameworks

Tool	HPC-Integration	Extra Infrastruktur	Wissenschaftliche Array-Unterstützung	Geeignet für diesen Bericht
Flink	niedrig/mittel	ja	begrenzt	nein
NATS	niedrig	ja	begrenzt	nein
Pulsar	niedrig/mittel	ja	begrenzt	nein
ADIOS2	hoch	nein	ja	ja

Die Entscheidung für ADIOS2 wird durch Tabelle 2 verdeutlicht. Im Gegensatz zu Flink, NATS oder Pulsar benötigt ADIOS2 keine zusätzliche Cluster-Infrastruktur. Es integriert sich nahtlos in HPC-Jobs und unterstützt Backends wie Buffer Protocol (BP), Hierarchical Data Format 5 (HDF5) und Message Passing Interface (MPI). Es ermöglicht speichereffiziente Datenübertragung und I/O mit niedriger Latenz, was besonders wichtig für große Simulationen ist. Für dieses Seminarprojekt ist ADIOS2 die optimale Wahl, da es als einziges Tool direkt für die HPC-Umgebung konzipiert ist und einen klaren, wissenschaftlich relevanten Vergleich zwischen batchbasierter Datenverarbeitung und Stream basierter Analyse am selben Anwendungsfall ermöglicht.

3.2 Plattformauswahl

Neben der Wahl des Datenframeworks ist die Entscheidung für die Ausführungsumgebung von zentraler Bedeutung. In der HPC-Praxis ist Slurm [slurm] der De-facto-Standard für Workload-Management. Slurm ist ein open source, fehlertoleranter und hochskalierbarer Workload-Manager [slurm], der weltweit für HPC-Cluster verwendet wird [JW23]. Slurm bietet eine robuste Job-Planung, Ressourcenverwaltung und Unterstützung für Batch-, interaktive und parallele Workloads auf HPC-Systemen [JW23]. Die Vorteile liegen in der Kompatibilität mit HPC-Systemen und ADIOS2 sowie der Möglichkeit, reproduzierbare, skalierbare und messbare Batch-Jobs zu erstellen [JW23]. Slurm integriert sich nahtlos in HPC-Infrastrukturen und gewährleistet, dass die Ergebnisse repräsentativ für reale wissenschaftliche Rechenworkflows sind [JW23].

Demgegenüber steht JupyterHPC [JupyterHPC], das Jupyter-Notebook-Umgebungen für HPC-Systeme erweitert und interaktive, codebasierte Arbeitsweisen mit direkter Visualisierung ermöglicht [SLB26]. Es eignet sich besonders gut für Prototypen, Debugging und die Erstellung von interaktiven Berichten [SLB26]. Die Vorteile liegen in der interaktiven Entwicklung mit sofortiger Rückmeldung, was ideal für das Testen, Debuggen

und Visualisieren von Algorithmen ist [SLB26]. Allerdings ist JupyterHPC nicht für Produktionsumgebungen oder langlaufende Batch-Jobs konzipiert und bietet eine geringe Unterstützung für echte Streaming-Workflows und Echtzeit-Datenpipelines. Zudem kann es durch Kernverwaltung und Server-Overhead zusätzliche Verzögerungen verursachen [SLB26].

JupyterHPC wurde für die Implementierung und Messung in diesem Projekt gewählt, da der Fokus auf der vergleichenden Analyse der Datenverarbeitungsmuster und nicht auf der Optimierung des Job-Schedulings lag. JupyterHPC ermöglicht eine direkte Rückmeldung während der Implementierung [SLB26], was für die Fehlersuche und die Validierung der Metriken im Rahmen eines Seminarprojekts von Vorteil ist. Zwar ist Slurm für langlaufende Produktionsjobs besser geeignet [JW23], doch für den Zweck der vergleichenden Analyse und der Demonstration der Konzepte innerhalb der Seminarzeit bietet JupyterHPC die notwendige Flexibilität und Transparenz. Diese Entscheidung gewährleistet, dass der Vergleich zwischen Batch- und Streaming-Verarbeitung in einer Umgebung durchgeführt wird, die den Fokus auf die Datenverarbeitung selbst legt, ohne durch komplexe Job-Scripting-Infrastrukturen abgelenkt zu werden. Die Ergebnisse sind dennoch aussagekräftig, da ADIOS2 als Abstraktionsschnittstelle die zugrundeliegende Infrastruktur weitgehend entkoppelt [ADIOS].

4 Methodik und Systemarchitektur

4.1 Das Simulationsmodell

Im Rahmen dieser Arbeit wird die Wärmediffusion auf einem 2D-Gitter als exemplarischer Anwendungsfall untersucht. Die Simulation modelliert die Ausbreitung von Wärme auf einem 2D-Gitter, wobei der Mittelpunkt zu Beginn heiß ist und der Rest des Gitters kalt startet. Wärmediffusion beschreibt den Prozess, bei dem sich Temperaturunterschiede mit der Zeit ausgleichen, indem Wärme von Bereichen hoher zu Bereichen niedriger Temperatur übertragen wird. Für numerische Simulationen wird dieser Prozess häufig mit der Finite-Differenzen-Methode diskretisiert, indem das Gebiet in ein Gitter unterteilt und die Temperatur an jedem Gitterpunkt schrittweise berechnet wird [Duf13].

Die Kernlogik der Simulation ist für Batch und Streaming identisch, um eine faire Vergleichbarkeit zu gewährleisten. Die Gittergrößen sind 100x100, 150x150 und 1500x1500 Pixel. Die Simulation läuft über 100 Zeitschritte, sodass eine Folge von Temperaturfeldern entsteht, die den zeitlichen Verlauf der Simulation repräsentieren. Jeder Zeitschritt erzeugt eine vollständige Temperaturmatrix, die entweder zunächst gesammelt und erst am Ende weiterverarbeitet oder unmittelbar nach der Berechnung übertragen wird. Der Anfangszustand wird durch einen quadratischen Bereich von 10x10 Pixeln im Zentrum definiert, der mit einem Wert von 100 initialisiert wird, während der Rest des Gitters mit 0 befüllt wird. Für jeden inneren Gitterpunkt wird die Temperatur mit der folgenden Update-Regel aus den vier direkten Nachbarn berechnet.

$$T_{i,j}^{n+1} = T_{i,j}^n + \alpha \cdot (T_{i-1,j}^n + T_{i+1,j}^n + T_{i,j-1}^n + T_{i,j+1}^n - 4T_{i,j}^n)$$

Die Zeitschrittgröße α ist auf 0,25 festgelegt. Diese Regel bewirkt, dass sich jeder Gitterpunkt schrittweise an den Temperaturwerten seiner direkten Umgebung orientiert. Dadurch glättet sich die Temperaturverteilung mit der Zeit, was das typische Verhalten einer Diffusion gut abbildet [Duf13].

Der Zweck der Simulation liegt nicht in komplexer physikalischer Modellierung, sondern in der Erzeugung strukturierter, zeitabhängiger Daten, um das I/O- und Verarbeitungsverhalten von Batch- und Streaming-Ansätzen zu vergleichen. Die Simulation ist bewusst einfach gehalten, da nicht die physikalische Modellierung im Vordergrund steht, sondern die Erzeugung strukturierter, zeitabhängiger Daten. Genau darin liegt auch die Eignung für den Vergleich von Batch- und Streaming-Verarbeitung. Die Implementierung erfolgt in reinem Python mit NumPy. Das Temperaturfeld wird als zweidimensionales NumPy-Array gespeichert und in jedem Zeitschritt mit Array-Slicing aktualisiert. Dies ermöglicht, dass der Fokus auf dem I/O-Verhalten und nicht auf der Rechengeschwindigkeit der Simulation selbst liegt.

4.2 Datenfluss und ADIOS2 Integration

Die Datenverarbeitung erfolgt über das ADIOS2-Framework, das als Schnittstelle zwischen der Simulationslogik und der Analyse dient. Im Batch-Modus werden alle Simulationsdaten vollständig berechnet und dabei in einer Datei gespeichert. Dies entspricht dem klassischen wissenschaftlichen Workflow, bei dem erst nach Abschluss der Rechnung die Analyse beginnt. Das Dateiformat ist eine ADIOS2-BP-Datei, wobei die Engine im Schreibmodus läuft. Die Datenstruktur besteht aus dem `temperature` 2D-Array (100x100, float64) und dem `step` Skalar (int32). Der Schreibzyklus beginnt mit `s.begin_step()`, gefolgt vom Schreiben der Temperatur und des Schrittzählers, und endet mit `s.end_step()`, das den Schritt in die Datei speichert. Alle Daten werden während der Simulation in einer Datei gesammelt, und der Consumer liest die Datei nachträglich, sobald die Produktion abgeschlossen ist.

Im Streaming-Modus werden die Daten während der Simulation Schritt für Schritt übertragen, ohne dass alle Daten im Speicher gehalten werden müssen. Dies ermöglicht eine parallele Verarbeitung durch den Consumer. Das Dateiformat und die Struktur sind identisch zur Batch-Variante, jedoch erfolgt nach `s.end_step()` eine sofortige Veröffentlichung des Schritts für den Consumer. Jeder Schritt wird unabhängig und zeitlich getrennt übertragen, sodass der Consumer sofort mit der Analyse des ersten Schritts beginnen kann, während der Producer noch an späteren Schritten rechnet. Die Kommunikation zwischen Producer und Consumer erfolgt über das Dateisystem-basierte Backend von ADIOS2 (BP Engine). Producer und Consumer sind logisch getrennt und laufen als separate Prozesse, die über die JupyterHPC-Umgebung gesteuert werden. ADIOS2 nutzt hier Puffermechanismen, um die Daten effizient zwischen den Prozessen zu bewegen, ohne dass ein explizites Netzwerk-Protokoll notwendig ist.

Der Consumer liest jede produzierte Temperaturmatrix und berechnet einfache Statistiken pro Zeitschritt. Die ADIOS2-Engine läuft im Lesemodus. Der Datenempfang erfolgt über `s.steps()`, welches eine Iterator-Struktur über alle Zeitschritte liefert, sowie `s.read('temperature')` und `s.read('step')` zum Auslesen der Daten. Die Analyseoperationen umfassen die Berechnung des Maximalwerts und des Durchschnittswerts. Diese Metriken werden in Listen gespeichert und dienen dazu, die Entwicklung des Simulationsergebnisses zu beobachten. Der Consumer empfängt die Daten über ADIOS2, berechnet die Statistiken und misst die Leistung hinsichtlich Latenz und Laufzeit. Die Visualisierung erfolgt direkt in der JupyterHPC-Umgebung, was eine sofortige Interpretation der Ergebnisse ermöglicht.

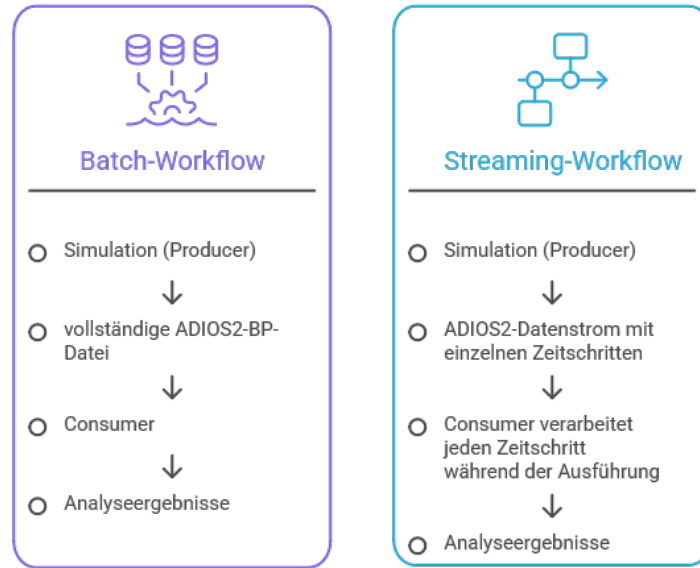


Abbildung 1: Workflow-Vergleich Batch vs. Streaming

4.3 Evaluierungsmetriken und Messmethodik

Um die Leistungsfähigkeit und Eignung von Batch- und Stream-Verarbeitung im Kontext der Wärmeleitungssimulation objektiv zu bewerten, werden mehrere quantitative und qualitative Metriken herangezogen. Diese Metriken ermöglichen einen klaren Vergleich zwischen den beiden Ansätzen und helfen, deren Stärken und Schwächen im Hinblick auf HPC-Umgebungen zu identifizieren.

Latenz bezeichnet in dieser Arbeit die Zeit bis zum ersten analysierbaren Ergebnis, nicht die Verarbeitungszeit des vollständigen Datensatzes. Konkret wird die Zeitspanne zwischen dem Eintreffen neuer Daten und dem Abschluss ihrer Verarbeitung [LCJ18], also die Zeit zwischen dem Versand des ersten Datenblocks (Schritt 0) durch den Producer und der Verfügbarkeit des ersten Ergebnisses durch den Consumer gemessen. Dies ist entscheidend, da im Batch-Modus der Consumer warten muss, bis die komplette Datei geschrieben ist, während im Stream-Modus der erste Schritt fast sofort verarbeitet werden kann. Niedrigere Latenz ist besser, da sie eine schnelle Reaktion auf dynamische Prozesse ermöglicht. In wissenschaftlichen Kontexten kann dies bedeuten, dass Simulationen frühzeitig abgebrochen werden können, wenn unerwünschte Ergebnisse auftreten. Basierend auf der Literatur [Eis+24], die eine signifikante Latenzreduktion durch ADIOS2-Streaming in HPC-Workflows dokumentiert, wird folgende Hypothese aufgestellt:

H1: Streaming reduziert die Latenz signifikant im Vergleich zu Batch-Processing.

Die Gesamtlaufzeit bezeichnet die Zeit vom Start des Prozesses bis zum Abschluss der Datenverarbeitung. Zur Messung erfolgt eine Zeitmessung mit `time.time()` am Anfang und Ende des jeweiligen Skripts (Producer und Consumer). Für Batch ist die Gesamtlaufzeit die Summe aus Producer und Consumer, da diese sequenziell arbeiten. Für Stream ist die Gesamtlaufzeit das Maximum aus Producer und Consumer, da diese parallel arbeiten. Kürzere Gesamtlaufzeit ist besser, da sie die Effizienz der Gesamtverarbeitung erhöht. Dies ist besonders relevant, wenn Rechenressourcen teuer sind oder wenn viele Simulationen hintereinander ausgeführt werden müssen. Die Literatur [Poe+22] zeigt, dass die Überlappung von Berechnung und I/O-Operationen zu erheblichen Zeitersparnissen führt. Daraus leitet sich folgende Hypothese ab:

H2: Stream Processing erreicht eine kürzere Gesamtlaufzeit als Batch-Processing durch die Parallelisierung von Berechnung und Analyse.

Skalierbarkeit beschreibt die Fähigkeit, mit zunehmender Datenmenge effizient zu arbeiten. Zur Messung wird die Latenz und die Gesamtlaufzeit bei steigender Gittergröße verglichen. Eine gute Skalierbarkeit liegt vor, wenn die Latenz und die Gesamtlaufzeit möglichst wenig zunehmen und die Verarbeitung somit auch für größere Datenmengen effizient bleibt. Dies ist entscheidend für die Anwendung auf reale wissenschaftliche Probleme, die oft Terabytes an Daten erzeugen. In der Literatur [Ihd+22] wird die "I/O-Wall" als Phänomen beschrieben, bei dem die Dateisystemleistung mit wachsender Datenmenge zum limitierenden Faktor wird. Basierend darauf wird folgende Hypothese formuliert:

H3: Batch-Processing verliert bei wachsender Datenmenge stärker an Leistung als Stream Processing.

Um statistische Signifikanz zu gewährleisten und zufällige Schwankungen durch Systemlast oder Hintergrundprozesse zu minimieren, wurden alle Benchmarks 10-mal wiederholt. Die in diesem Bericht präsentierten Werte stellen den arithmetischen Mittelwert dieser 10 Durchläufe dar. Zusätzlich wurde die Standardabweichung (SD) berechnet, um die Variabilität der Messergebnisse zu quantifizieren. Diese Vorgehensweise erhöht die Zuverlässigkeit der Messergebnisse und stellt sicher, dass die beobachteten Unterschiede zwischen Batch und Streaming nicht auf zufällige Ergebnisse zurückzuführen sind.

Zur Sicherstellung der Reproduzierbarkeit und Transparenz werden die technischen Details der experimentellen Umgebung hier spezifiziert. Die Implementierung erfolgte in Python 3.12 unter Verwendung der Bibliotheken NumPy 2.3 und ADIOS2 2.12. Die Ausführung fand in der JupyterHPC-Umgebung auf einem standardisierten Knoten des HPC-Clusters der Universität Göttingen statt. Producer und Consumer wurden als separate Prozesse auf demselben Knoten ausgeführt. Als Dateisystem-Backend wurde ADIOS2 BP verwendet, das auf dem parallelen Dateisystem des Clusters gespeichert wurde. Um den Einfluss von Dateisystem-Caching auf die Batch-Ergebnisse zu minimieren, wurde vor jedem Messlauf sichergestellt, dass keine verbleibenden Cache-Daten aus vorherigen Läufen die Zeiten verfälschen, indem die Dateien nach jedem Lauf gelöscht wurden.

5 Ergebnisse

5.1 Latenzanalyse

Die Latenz, definiert als die Zeit bis zum ersten analysierbaren Ergebnis, zeigt den deutlichsten Unterschied zwischen den beiden Verarbeitungsmodellen. Tabelle 3 fasst die gemessenen Latenzwerte für die verschiedenen Gittergrößen zusammen.

Tabelle 3: Vergleich der Latenz (in Sekunden)

Gittergröße	Modell	Mittelwert	SD
100x100	Batch	11,132	0,443
	Stream	0,460	0,462
150x150	Batch	11,300	0,413
	Stream	0,184	0,295
1500x1500	Batch	32,713	0,999
	Stream	0,440	0,494

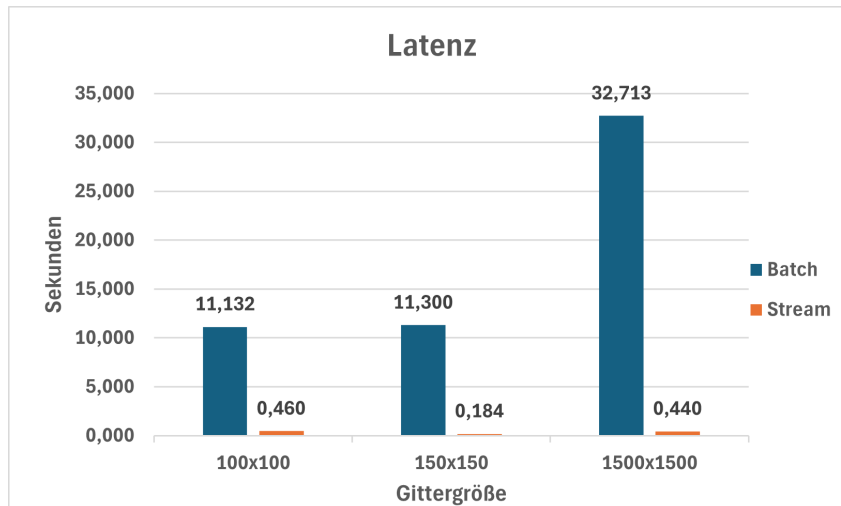


Abbildung 2: Latenzvergleich

Im Batch-Modus ist die Latenz direkt an die Laufzeit des Producers gekoppelt, da der Consumer erst nach dem vollständigen Schreibvorgang der Datei beginnen kann. Bei der größten Gittergröße (1500x1500) beträgt die Latenz hier 32,713 Sekunden. Im Gegensatz dazu bleibt die Latenz im Stream-Modus über alle Gittergrößen hinweg stabil und niedrig, mit einem Maximum von 0,460 Sekunden bei der kleinsten und 0,440 Sekunden bei der größten Gittergröße. Dies entspricht einer Reduktion der Latenz um bis zu 98,7 %. Die SD-Werte in Tabelle 3 zeigen, dass die Messungen trotz der hohen Variabilität im Batch-Modus bei großen Daten (SD 0,999) reproduzierbar sind, während der Streaming-Modus eine sehr stabile Latenz aufweist. Wie in Abbildung 2 visualisiert, wird der massive Vorteil des Streamings bei steigender Datenmenge besonders deutlich. Diese Ergebnisse bestätigen die theoretische Annahme, dass Stream Processing durch die Vermeidung von Wartezeiten auf Dateiabschlüsse Echtzeit-Einblicke ermöglicht [Eis+24].

5.2 Laufzeitanalyse

Die Gesamtlaufzeit des Workflows variiert in Abhängigkeit von der Datenmenge. Wie in Tabelle 4 dargestellt, ist bei kleinen Datenmengen (100x100 und 150x150) das Batch-Verfahren effizienter.

Tabelle 4: Vergleich der Gesamtlaufzeit (in Sekunden)

Gittergröße	Modell	Mittelwert	SD
100x100	Batch	0,445	0,097
	Stream	0,545	0,518
150x150	Batch	0,627	0,123
	Stream	0,872	0,459
1500x1500	Batch	39,656	0,905
	Stream	22,493	0,909

Bei der Gittergröße 100x100 ist Batch-Processing mit 0,445 Sekunden etwa 18 % schneller als Stream Processing (0,545 Sekunden). Dies liegt daran, dass bei kleinen Datenmengen der Streaming-Overhead (Initialisierung, Schrittverwaltung) die Vorteile der Parallelität überwiegt. Dieser Trend kehrt sich jedoch bei großen Datenmengen um. Bei

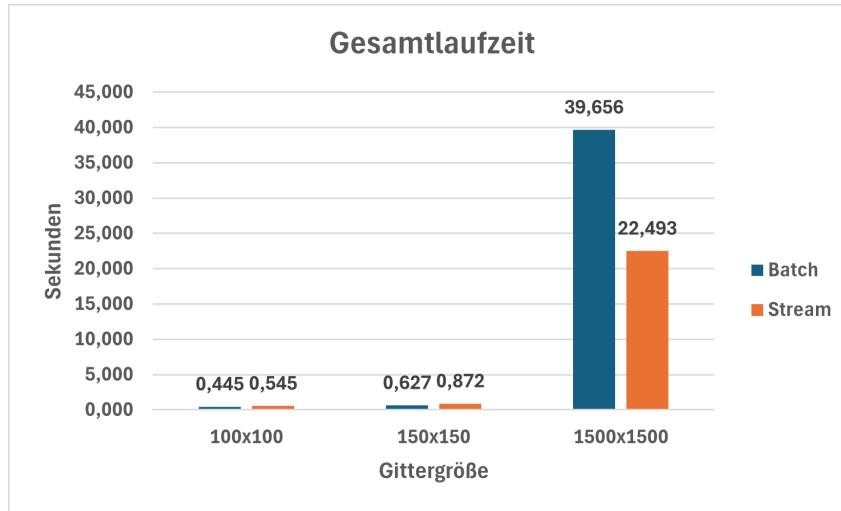


Abbildung 3: Gesamtlaufzeitvergleich

1500x1500 Pixeln beträgt die Gesamtlaufzeit im Batch-Modus 39,656 Sekunden, während Stream Processing den Workflow in 22,493 Sekunden abschließt. Dies entspricht einer Zeitersparnis von ca. 43 % zugunsten des Streaming-Ansatzes. Basierend auf den getesteten Gittergrößen liegt der Break-Even-Point, an dem die Vorteile der Parallelität die Initialisierungs-Overheads des Streamings kompensieren, zwischen 150x150 und 1500x1500 Pixeln. Um diesen Punkt jedoch präziser zu bestimmen, wären zusätzliche Tests mit intermediate Gittergrößen erforderlich. Die SD-Werte in Tabelle 4 deuten darauf hin, dass die Laufzeitvariabilität bei Streaming mit kleineren Datenmengen höher ist (SD 0,518), was auf einen Initialisierungs-Overhead zurückzuführen sein könnte, während sie bei großen Datenmengen stabil bleibt. Abbildung 3 verdeutlicht diesen Break-Even-Point, indem der Schnittpunkt der Effizienz zwischen den beiden Modellen bei den mittleren Datenmengen liegt.

5.3 Skalierbarkeit und Komponentenanalyse

Um die Ursachen für die Unterschiede in der Gesamtlaufzeit und das Skalierungsverhalten zu verstehen, wurden die Laufzeiten der einzelnen Komponenten Producer (Simulation) und Consumer (Analyse) über alle Gittergrößen hinweg betrachtet. Tabelle 5 stellt diese Werte im Vergleich gegenüber.

Tabelle 5: Komponentenanalyse der Laufzeiten (in Sekunden)

Gittergröße	Modell	Producer	SD	Consumer	SD
100x100	Batch	0,286	0,081	0,158	0,035
	Stream	0,286	0,081	0,545	0,518
150x150	Batch	0,367	0,047	0,261	0,091
	Stream	0,380	0,075	0,872	0,459
1500x1500	Batch	21,612	0,823	18,043	0,265
	Stream	22,075	0,895	22,493	0,909

Die Producer-Laufzeit ist in beiden Modi über alle Größen nahezu identisch (Abweichung < 3 %), was bestätigt, dass die Simulationslogik unabhängig vom I/O-Modell konsistent arbeitet. Die SD der Producer-Zeiten ist in allen Fällen gering, was auf eine

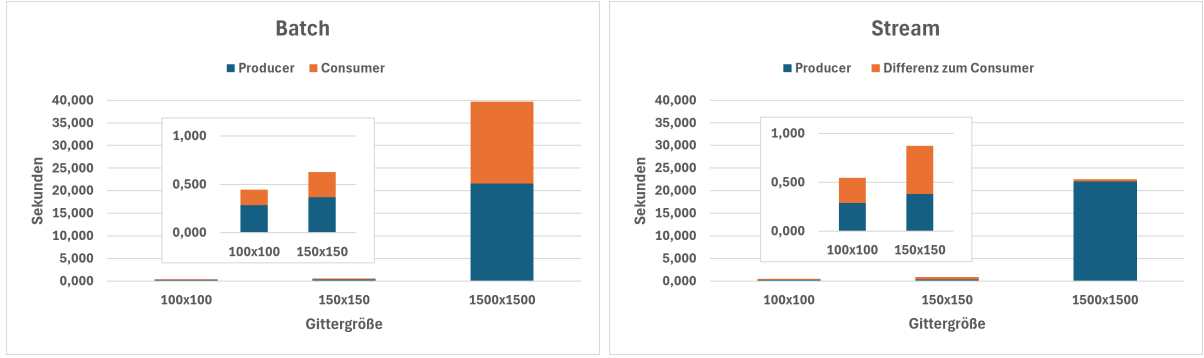


Abbildung 4: Komponentenanalyse

stabile Simulationsausführung hindeutet. Auffällig ist jedoch das Verhalten des Consumers: Im Batch-Modus ist die reine Consumer-Laufzeit stets niedriger als im Stream-Modus (z. B. 18,043 s vs. 22,493 s bei 1500x1500). Dies verdeutlicht, dass der Streaming-Consumer selbst ineffizienter sein kann als der Batch Consumer, aber der gesamte Workflow schneller ist, weil Producer und Consumer parallel laufen. Im Batch-Verfahren kann der Consumer die vollständige Datei sequenziell und ohne Synchronisationspunkte einlesen. Im Stream-Modus entsteht durch die Schritt-für-Schritt-Übertragung und die Pufferung in ADIOS2 ein gewisser Overhead pro Zeitschritt, der die reine Analysezeit erhöht. Dennoch gewinnt Stream Processing bei großen Datenmengen aufgrund der Berechnungslogik der Gesamtlaufzeit. Während im Batch-Modus die Zeiten addiert werden ($T_{gesamt} = T_{prod} + T_{cons}$), entspricht sie im Stream-Modus dem Maximum der beiden Zeiten ($T_{gesamt} = \max(T_{prod}, T_{cons})$), da Berechnung und Analyse überlappen [Poe+22]. Bei kleinen Gittergrößen (100x100, 150x150) dominiert der Streaming-Overhead des Consumers die Gesamtzeit, da der Producer sehr schnell fertig ist und der Consumer langsamer arbeitet als im Batch-Modus. Bei der großen Gittergröße (1500x1500) gleichen sich Producer- und Consumer-Zeiten im Stream-Modus an (22,075 s vs. 22,493 s), wodurch die Parallelität maximal wirksam wird. Abbildung 4 macht diese Überlappung sichtbar, indem sie zeigt, dass die Consumer-Zeit im Stream-Modus zwar höher ist, aber nicht zur Gesamtzeit addiert wird. Hinsichtlich der Skalierbarkeit zeigt sich zudem, dass die Latenz im Batch-Modus linear mit der Datenmenge ansteigt, da größere Dateien länger geschrieben werden müssen. Im Stream-Modus bleibt die Latenz hingegen weitgehend konstant (siehe Tabelle 3), da sie primär von der Übertragungszeit eines einzelnen Zeitschritts abhängt. Dies deutet darauf hin, dass Stream Processing besser gegen die sogenannte "I/O-Wall" skaliert, bei der das Dateisystem zum limitierenden Faktor wird [Ihd+22]. Allerdings basieren diese Skalierbarkeitsbeobachtungen auf nur drei getesteten Gittergrößen. Während das größte Gitter (1500x1500) signifikant größer ist, repräsentiert es möglicherweise nicht die extremen Datenmengen realer großskaliger HPC-Simulationen. Um robustere Schlussfolgerungen zur Skalierbarkeit zu ziehen, wären weitere Größenstufen und idealerweise Mehrknoten-Experimente erforderlich.

6 Diskussion

Die in Kapitel 5 präsentierten Ergebnisse ermöglichen eine differenzierte Einordnung der beiden Verarbeitungsparadigmen im Kontext wissenschaftlicher HPC-Workflows. Die Dis-

kussion gliedert sich in die Interpretation der Leistungsvorteile, die praktischen Implikationen für die Workflow-Planung sowie die Grenzen der durchgeführten Studie.

6.1 Interpretation der Leistungsvorteile

Die massive Latenzreduktion im Stream-Modus validiert die Hypothese H1, dass Streaming für Szenarien mit Echtzeitanforderungen überlegen ist. Dies ist besonders relevant für Anwendungen, bei denen Simulationen frühzeitig abgebrochen oder parametrisiert werden müssen, wenn Zwischenergebnisse bestimmte Kriterien verletzen [Eis+24]. Die Möglichkeit zur In-situ-Analyse, bei der Daten im Speicher verarbeitet werden, ohne sie persistent auf ein Dateisystem zu schreiben, umgeht den Engpass des parallelen Dateisystems, der in modernen HPC-Systemen zunehmend zur "I/O-Wall" führt [Ali+25]. Der scheinbare Widerspruch, dass der Consumer im Stream-Modus langsamer arbeitet, der Gesamtworkflow jedoch bei großen Datenmengen schneller ist, lässt sich durch die Ressourcennutzung erklären. Batch-Processing erzwingt eine sequenzielle Abarbeitung, was zu Leerlaufzeiten führt: Der Consumer wartet auf den Producer, oder Ressourcen liegen brach, während auf I/O-Operationen gewartet wird. Stream Processing entkoppelt diese Prozesse. Der Übergang von dateibasierten zu Streaming-Pipelines ermöglicht eine bessere Auslastung der Rechenkerne, da I/O und Berechnung überlappt werden können [Poe+22]. Der Overhead, der bei kleinen Datenmengen zu Nachteilen für Streaming führt, resultiert aus der Initialisierung der ADIOS2-Engine und dem Management der Datenströme, was bei kurzen Laufzeiten ins Gewicht fällt [LCJ18]. Wichtig ist: Streaming verbessert die Gesamtlaufzeit nur, wenn der Nutzen der Überlappung größer ist als der Streaming-Overhead.

6.2 Praktische Implikationen für HPC-Workflows

Basierend auf den Ergebnissen lassen sich konkrete Empfehlungen für die Auswahl des Verarbeitungsmodells ableiten, wobei die Datenmenge als primärer Entscheidungsfaktor fungiert. Für kleine, statische Datensätze oder schnelle Testläufe bleibt Batch-Processing aufgrund seiner Einfachheit und des geringeren Overheads die effizientere Wahl [Vin24]. Der Aufwand für die Einrichtung einer Streaming-Pipeline lohnt sich erst ab einer bestimmten Datenschwelle, die in diesem Experiment zwischen 150x150 und 1500x1500 Pixeln lag, wobei weitere Tests mit intermediären Größen die Genauigkeit dieser Schätzung erhöhen würden. Unterhalb dieser Schwelle überwiegen die Kosten für die Pipeline-Initialisierung den Nutzen der Parallelität.

Neben der Datenmenge sind die Latenzanforderungen entscheidend. Wenn wissenschaftliche Erkenntnisse zeitkritisch sind, beispielsweise bei der Steuerung von Experimenten oder dem Monitoring von Langzeitsimulationen, ist Stream Processing unverzichtbar, unabhängig von der Gesamtlaufzeit. Die niedrige Latenz ermöglicht frühere Entscheidungen. Für Produktionsumgebungen, in denen sowohl Echtzeit-Monitoring als auch langfristige Datensicherheit erforderlich sind, erscheinen hybride Ansätze vielversprechend. Dabei könnten kritische Metriken gestreamt werden, während die vollständigen Rohdaten asynchron im Batch-Verfahren zur Archivierung geschrieben werden. Dies würde die Vorteile der niedrigen Latenz mit der Persistenz und Reproduzierbarkeit von Batch-Dateien verbinden. Die Entscheidung ist somit immer kontextabhängig und muss die spezifischen Anforderungen des Workflows bezüglich Geschwindigkeit, Sicherheit und Infrastruktur berücksichtigen.

6.3 Grenzen der Studie

Bei der Interpretation der Ergebnisse müssen bestimmte Einschränkungen berücksichtigt werden.

Erstens wurde nur ein einziges Simulationsmodell verwendet (2D-Wärmediffusion). Die Simulation ist physikalisch leichtgewichtig und repräsentiert nicht die Komplexität realer wissenschaftlicher Anwendungen. In komplexeren Szenarien könnte das Verhältnis von Rechenzeit zu I/O-Zeit verschoben sein, was die Ergebnisse beeinflusst.

Zweitens erfolgte die Evaluierung in einer JupyterHPC-Umgebung und nicht in einem vollständigen Slurm-Produktionsworkflow. Obwohl ADIOS2 die Infrastruktur weitgehend abstrahiert, kann der Overhead durch die Kernverwaltung und den Server von Jupyter die absoluten Laufzeiten im Vergleich zu einer Slurm-Umgebung erhöhen [SLB26]. Die relativen Unterschiede zwischen Batch und Stream sollten jedoch valide bleiben.

Drittens wurden zur Skalierbarkeitsanalyse nur drei Gittergrößen verwendet. Obwohl dies für eine Seminararbeit akzeptabel ist, limitiert dies die Generalisierbarkeit der Ergebnisse auf großen Datensätzen. Zudem wurden die Experimente auf einem einzelnen Knoten durchgeführt. Mehrknoten-Experimente wären notwendig, um die Skalierbarkeit unter realen Netzwerkbedingungen und verteiltem Dateisystemzugriff vollständig zu validieren.

Viertens wurde die Fehlertoleranz und Datenpersistenz nicht tiefgehend verglichen. Stream Processing ohne persistente Zwischenspeicherung birgt das Risiko des Datenverlusts bei Systemabstürzen, was durch Checkpoint-Mechanismen adressiert werden müsste [LCJ18]. Batch-Dateien bieten hier eine höhere Robustheit durch Persistenz auf dem Dateisystem, was für langfristige Datensicherheit entscheidend sein kann.

Fünftens könnte Dateisystem-Caching die Batch-Messungen beeinflussen. Obwohl vor jedem Messlauf sichergestellt wurde, dass keine verbleibenden Cache-Daten aus vorherigen Läufen die Zeiten verfälschen, bleibt die Möglichkeit bestehen, dass Caching-Mechanismen des Dateisystems die Lesezeiten im Batch-Modus verbessern und somit den Vorteil des Streamings relativieren.

Sechstens ist die Consumer-Analyse bewusst einfach gehalten (Berechnung von Maximum und Durchschnitt). Komplexere Analysen, wie etwa maschinelles Lernen oder aufwendige Visualisierungen, könnten das Verhältnis von Verarbeitungszeit zu I/O-Zeit verändern und somit den Break-Even-Point verschieben.

Zusammenfassend zeigt die Untersuchung, dass Stream Processing kein universeller Ersatz für Batch-Processing ist, sondern eine spezialisierte Ergänzung für datenintensive und latenzkritische Workflows. Die Tendenz in der HPC-Landschaft geht jedoch klar dahin, Streaming-Technologien wie ADIOS2 zu integrieren, um der wachsenden Diskrepanz zwischen Rechenleistung und I/O-Bandbreite zu begegnen.

7 Fazit

Diese Arbeit hat den Vergleich von Streaming Data Analytics und Batch-Processing in HPC-Systemen anhand einer Wärmediffusionssimulation durchgeführt. Die zentrale Forschungsfrage, wie sich die beiden Verarbeitungsmodelle unterscheiden, konnte durch quantitative Messungen beantwortet werden.

Stream Processing ist vorteilhaft für große, datenintensive und latenzkritische HPC-Workflows, da es den frühen Zugriff auf Zwischenergebnisse ermöglicht und Berechnung

mit Analyse überlagern kann. Es ist jedoch kein universeller Ersatz für Batch-Processing. Die Latenzreduktion um bis zu 98 % durch Stream Processing ist ein signifikanter Vorteil, der in klassischen Batch-Workflows nicht erreichbar ist. Bei großen Datenmengen wird eine bis zu 43 % schnellere Gesamtlaufzeit durch die Überlappung von Berechnung und Analyse erreicht. Dies macht Streaming zur überlegenen Wahl, wenn die I/O-Last den Engpass darstellt.

Batch-Processing bleibt jedoch sinnvoll für kleine Workloads, einfache Nachbearbeitungsaufgaben und Szenarien, in denen Persistenz und Reproduzierbarkeit wichtiger sind als niedrige Latenz. Es wird empfohlen, Stream Processing für Echtzeit-Analysen und große Datenmengen zu wählen, insbesondere wenn die "I/O-Wall" vermieden werden muss. In der Praxis können hybride Ansätze am besten geeignet sein, indem Streaming für das Echtzeit-Monitoring mit Batch zur Archivierung kombiniert wird. Ein Risiko von Stream Processing ist der Datenverlust bei Abbruch (keine persistente Datei) und eine höhere Komplexität der Infrastruktur.

Tabelle 6: Zusammenfassung der Handlungsempfehlungen

Szenario	Empfohlener Ansatz
Kleine Testläufe	Batch
Große Simulationsausgaben	Streaming
Schneller Zugriff auf erste Ergebnisse	Streaming
Robuste persistente Ausgabe	Batch
Monitoring und Archivierung	Hybrid
Echtzeit-Monitoring	Streaming
Offline-Nachbearbeitung	Batch

Zukünftige Arbeiten sollten hybride Ansätze untersuchen, die die Vorteile beider Welten kombinieren. Zudem wäre eine Untersuchung auf echten Supercomputern mit verteiltem Dateisystem ratsam, um die Skalierbarkeit unter realen Netzwerkbedingungen zu validieren. Die Wahl des Verarbeitungsmodells sollte nicht pauschal getroffen werden, sondern sich an den spezifischen Anforderungen des Workflows bezüglich Datenvolumen, Latenz und Datensicherheit orientieren.

Literatur

- [ADIOS] *Introduction — ADIOS2 2.12.1 documentation*. URL: <https://adios2.readthedocs.io/en/v2.12.1/introduction/introduction.html#what-adios2-is-and-isn-t> (besucht am 07.05.2026).
- [Ali+25] Arshad Ali u. a. „The Role of High Performance Computing Architectures“. In: *Spectrum of Engineering Sciences* (2. März 2025), S. 885–908. ISSN: 3007-3138.
- [Bel+20] S D Belov u. a. „BATCH AND STREAM BIG DATA PROCESSING PLATFORM: CASE OF NETWORK TRAFFIC ANALYSIS“. In: (2020).
- [Duf13] Daniel J. Duffy. *Finite Difference Methods in Financial Engineering: A Partial Differential Equation Approach*. Google-Books-ID: VVjjAQAAQ-BAJ. John Wiley & Sons, 28. Okt. 2013. 452 S. ISBN: 978-1-118-85648-2.
- [Eis+24] Greg Eisenhauer u. a. „Streaming Data in HPC Workflows Using ADIOS“. In: *Proceedings of the Cray User Group*. CUG 2024: Cray User Group. Perth Australia: ACM, 5. Mai 2024, S. 31–43. ISBN: 979-8-4007-1328-6. DOI: 10.1145/3725789.3725793.
- [flink] *Apache Flink Documentation*. URL: <https://nightlies.apache.org/flink/flink-docs-release-2.2/> (besucht am 07.05.2026).
- [Fre+23] Erick Fredj u. a. *accelerating wrf i/o performance with adios2 and network-based streaming*. 13. Apr. 2023. DOI: 10.48550/arXiv.2304.06603. arXiv: 2304.06603[cs].
- [Ihd+22] Nina Ihde u. a. „A Survey of Big Data, High Performance Computing, and Machine Learning Benchmarks“. In: *Performance Evaluation and Benchmarking*. Hrsg. von Raghunath Nambiar und Meikel Poess. Bd. 13169. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, S. 98–118. ISBN: 978-3-030-94436-0 978-3-030-94437-7. DOI: 10.1007/978-3-030-94437-7_7.
- [JupyterHPC] *JupyterHPC : Documentation for HPC*. URL: <https://docs.hpc.gwdg.de/services/jupyterhpc/index.html> (besucht am 11.05.2026).
- [JW23] Morris A. Jette und Tim Wickberg. „Architecture of the Slurm Workload Manager“. In: *Job Scheduling Strategies for Parallel Processing*. Hrsg. von Dalibor Klusáček, Julita Corbalán und Gonzalo P. Rodrigo. Bd. 14283. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2023, S. 3–23. ISBN: 978-3-031-43942-1 978-3-031-43943-8. DOI: 10.1007/978-3-031-43943-8_1.
- [LCJ18] Andre Luckow, George Chantzialexiou und Shantenu Jha. *Pilot-Streaming: A Stream Processing Framework for High-Performance Computing*. 11. Nov. 2018. DOI: 10.48550/arXiv.1801.08648. arXiv: 1801.08648[cs].
- [NATS25] *JetStream / NATS Docs*. 19. Dez. 2025. URL: <https://docs.nats.io/nats-concepts/jetstream> (besucht am 07.05.2026).

- [Poe+22] Franz Poeschel u. a. „Transitioning from File-Based HPC Workflows to Streaming Data Pipelines with openPMD and ADIOS2“. In: *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*. Hrsg. von Jeffrey Nichols u. a. Bd. 1512. Series Title: Communications in Computer and Information Science. Cham: Springer International Publishing, 2022, S. 99–118. ISBN: 978-3-030-96497-9 978-3-030-96498-6. DOI: 10.1007/978-3-030-96498-6.
- [Pulsar] *Pulsar Overview / Apache Pulsar*. URL: <https://pulsar.apache.org/docs/next/concepts-overview/> (besucht am 07.05.2026).
- [SLB26] Md Saeed Siddik, Hao Li und Cor-Paul Bezemer. „A systematic literature review of software engineering research on Jupyter notebook“. In: *Journal of Systems and Software* 235 (Mai 2026), S. 112758. ISSN: 01641212. DOI: 10.1016/j.jss.2025.112758.
- [slurm] *Slurm Workload Manager - Overview*. URL: <https://slurm.schedmd.com/overview.html> (besucht am 11.05.2026).
- [Vin24] Santosh Vinnakota. „Combining Batch and Stream Processing for Hybrid Data Workflows“. In: *International Journal on Science and Technology* 15.1 (7. Feb. 2024), S. 2819. ISSN: 2229-7677. DOI: 10.71097/IJSAT.v15.i1.2819.

A Batch Processing Producer Code

```

1  # Start timing
2  start_time = time.time()
3
4  # Simulation parameters
5  nx, ny = 100, 100
6  steps = 100
7  alpha = 0.25
8
9  # Initial grid
10 current = np.zeros((nx, ny), dtype=float)
11 current[50:60, 50:60] = 100.0
12
13 # Opening ADIOS2 engine for batch writing
14 with ad2.Stream('heat_batch.bp', 'w') as s:
15     for step in range(steps + 1):
16         # Skip diffusion step for initial condition
17         if step > 0:
18             new = current.copy()
19             new[1:-1, 1:-1] = current[1:-1, 1:-1] + alpha * (
20                 current[0:-2, 1:-1] + current[2:, 1:-1] +
21                 current[1:-1, 0:-2] + current[1:-1, 2:] -
22                 4 * current[1:-1, 1:-1]
23             )
24             current = new
25
26         # Write step
27         s.begin_step()
28         s.write('temperature', current)
29         s.write('step', np.array(step, dtype=np.int32))
30         s.end_step()
31
32         if step == 0:
33             first_step_sent = time.time()
34
35 # End timing
36 total_runtime = time.time() - start_time

```

B Stream Processing Producer Code

```

1  # Start timing
2  start_time = time.time()
3
4  # Simulation parameters
5  nx, ny = 100, 100
6  steps = 100
7  alpha = 0.25
8
9  # Initial grid
10 current = np.zeros((nx, ny), dtype=float)
11 current[50:60, 50:60] = 100.0
12
13 # Opening stream for writing
14 with ad2.Stream('heat_stream.bp', 'w') as s:
15     for step in range(steps + 1):
16         # Skip diffusion step for initial condition
17         if step > 0:
18             new = current.copy()
19             new[1:-1, 1:-1] = current[1:-1, 1:-1] + alpha * (
20                 current[0:-2, 1:-1] + current[2:, 1:-1] +
21                 current[1:-1, 0:-2] + current[1:-1, 2:] -
22                 4 * current[1:-1, 1:-1]
23             )
24             current = new
25
26         # Write step
27         s.begin_step()
28         s.write('temperature', current)
29         s.write('step', np.array(step, dtype=np.int32))
30         s.end_step()
31
32         if step == 0:
33             first_step_sent = time.time()
34
35 # End timing
36 total_runtime = time.time() - start_time

```

C Batch Processing Consumer Code

```
1  # Start timing
2  start_time = time.time()
3
4  # Open ADIOS2 stream for reading
5  with ad2.Stream('heat_batch.bp', 'r') as s:
6      step_count = 0
7      for step in s.steps():
8          # Read data
9          temperature = s.read('temperature')
10         step_num = s.read('step')
11
12         # Compute and store metrics
13         max_temp = np.max(temperature)
14         avg_temp = np.mean(temperature)
15         max_values.append(max_temp)
16         avg_values.append(avg_temp)
17
18         # First result latency
19         if step_count == 0:
20             first_step_received = time.time()
21
22         step_count += 1
23
24  # End timing
25  total_runtime = time.time() - start_time
```

D Stream Processing Consumer Code

```

1  # Open ADIOS2 stream for reading
2  start_wait = time.time()
3  while not os.path.exists('heat_stream.bp'):
4      time.sleep(0.1)
5  waiting_time = time.time() - start_wait
6
7  # Start timing
8  start_time = time.time()
9
10 with ad2.Stream('heat_stream.bp', 'r') as s:
11     step_count = 0
12     for step in s.steps():
13         # Read data
14         temperature = s.read('temperature')
15         step_num = s.read('step')
16
17         # Compute and store metrics
18         max_temp = np.max(temperature)
19         avg_temp = np.mean(temperature)
20         max_values.append(max_temp)
21         avg_values.append(avg_temp)
22
23         # First result latency
24         if step_count == 0:
25             first_step_received = time.time()
26
27         step_count += 1
28
29 # End timing
30 total_runtime = time.time() - start_time

```