



Annika Marie Günther

Streaming Data Analytics vs. Batch Processing in HPC-Systemen

Supervisor: Sepehr Mahmoodian

Warum solltest du dich für Stream Processing in HPC interessieren?

*Latenzreduktion von **32,713 Sekunden** (Batch) auf **0,440 Sekunden** (Stream)*

... aber wann lohnt sich Stream Processing gegenüber klassischem Batch Processing in HPC-Workflows?

Agenda

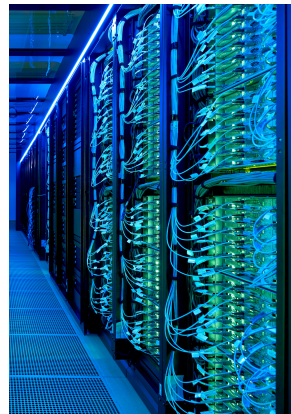
- 1 Problem
- 2 Hintergrund
- 3 Methodik
- 4 Ergebnisse
- 5 Diskussion
- 6 Fazit

Agenda

- 1 Problem
- 2 Hintergrund
- 3 Methodik
- 4 Ergebnisse
- 5 Diskussion
- 6 Fazit

HPC Datenflut

- HPC erzeugt große Datenmengen (Terabyte)
- Problem: I/O-Wall (Dateisystem wird zum Engpass)
- Folge: Rechenleistung wartet auf Schreibvorgänge



Ihde et al., "A Survey of Big Data, High Performance Computing, and Machine Learning Benchmarks", 2022; Ali et al., "The Role of High Performance Computing Architectures", 2025, Image source: <https://www.scc.kit.edu/forschung/4942.php>

Agenda

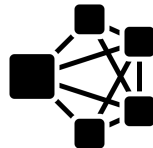
- 1 Problem
- 2 Hintergrund**
- 3 Methodik
- 4 Ergebnisse
- 5 Diskussion
- 6 Fazit

Was ist HPC?

■ **Definition:** Parallele Rechnersysteme (Cluster, Supercomputer)

■ **Merkmale:**

- ▶ Hohe Rechenleistung
- ▶ großer Speicher
- ▶ effiziente Knoten-Kommunikation



Ihde et al., "A Survey of Big Data, High Performance Computing, and Machine Learning Benchmarks", 2022; Ali et al., "The Role of High Performance Computing Architectures", 2025; Poeschel et al., "Transitioning from File-Based HPC Workflows to Streaming Data Pipelines with openPMD and ADIOS2", 2022, Image source:
https://www.flaticon.com/de/kostenloses-icon/cluster_6601301

Was ist Batch Processing

- Verarbeitet abgeschlossene, endliche Einheiten
- Vorteil: Einfach, etabliert in HPC
- Nachteil: Hohe Latenz, I/O-Last auf Dateisystem



Luckow, Chantzialexiou, and Jha, *Pilot-Streaming*, 2018; Vinnakota, "Combining Batch and Stream Processing for Hybrid Data Workflows", 2024; Ali et al., "The Role of High Performance Computing Architectures", 2025, Image source: https://www.flaticon.com/de/kostenloses-icon/dateiordner_18219605

Was ist Stream Processing

- Verarbeitet kontinuierliche Datenströme
- Vorteil: Niedrige Latenz, In-Situ-Analyse möglich
- Nachteil: Höhere Komplexität, Zustandsverwaltung nötig



Luckow, Chantzialexiou, and Jha, *Pilot-Streaming*, 2018; Eisenhauer et al., "Streaming Data in HPC Workflows Using ADIOS", 2024, Image source:

https://www.flaticon.com/de/kostenlos-icon/wasserhahn_7294849?term=wasserhahn&related_id=7294849

Batch vs. Stream

Kriterium	Batch Processing	Stream Processing
Latenz	Hoch	Niedrig
Infrastruktur	Standard	Spezialisiert
Komplexität	Einfach	Hoch
Anwendungsfall	Offline-Analyse	Echtzeit-Monitoring
Datenvolumen	Endlich	Unbegrenzt
Fehlertoleranz	Hoch	Niedriger (keine persistente Zwischenspeicherung)

Luckow, Chantzialexiou, and Jha, *Pilot-Streaming*, 2018; Vinnakota, “Combining Batch and Stream Processing for Hybrid Data Workflows”, 2024; Eisenhauer et al., “Streaming Data in HPC Workflows Using ADIOS”, 2024

Agenda

- 1 Problem
- 2 Hintergrund
- 3 Methodik**
- 4 Ergebnisse
- 5 Diskussion
- 6 Fazit

Toolauswahl Marktüberblick



Apache Flink

- Open-Source Streaming-Engine
- Cloud-Fokus



NATS JetStream

- Leichtgewichtiges Nachrichtensystem
- geringe Latenz



Apache Pulsar

- Verteiltes Messaging-System
- Cloud-Fokus



ADIOS2

- HPC-optimiert
- effiziente Datenbewegung

Apache Flink Documentation; JetStream | NATS Docs, 2025; Pulsar Overview | Apache Pulsar; Introduction — ADIOS2 2.12.1 documentation

Toolauswahl - Warum ADIOS2?

Anforderungen an das Tool

- **HPC-Integration:** Nahtlose Einbindung in bestehende HPC-Jobs
- **Infrastruktur:** Keine zusätzliche Cluster-Infrastruktur
- **Performance:** Niedrige Latenz bei In-situ/In-transit Datenbewegung

⇒ ADIOS2 erfüllt alle Anforderungen



Plattformauswahl - JupyterHPC

- Ermöglicht interaktive, codebasierte Arbeitsweise mit direkter Visualisierung
- Gut für Prototypen, Debugging und Dokumentation
- Nachteil: Overhead durch Kernel-Management



JupyterHPC; , Siddik, Li, and Bezemer, "A systematic literature review of software engineering research on Jupyter notebook", 2026

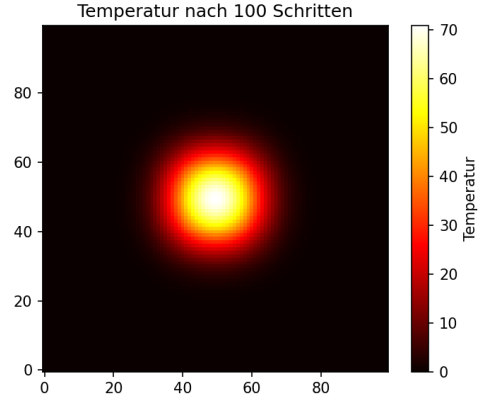
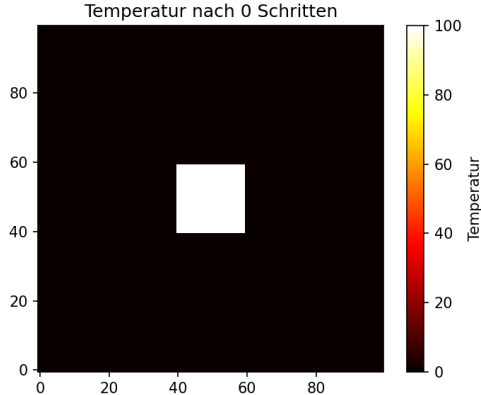
Simulationsaufbau - Physik & Prozess

- **Physik:** Numerische Approximation der Wärmeleitungsgleichung
- **Formel:** $T_{i,j}^{n+1} = T_{i,j}^n + \alpha \cdot (T_{i-1,j}^n + T_{i+1,j}^n + T_{i,j-1}^n + T_{i,j+1}^n - 4T_{i,j}^n)$ mit $\alpha = 0,25$
- **Prozess:**
 - ▶ Initialisierung eines heißen Kerns (10x10 Pixel im Zentrum)
 - ▶ Iterative Berechnung der Temperaturverteilung über 100 Zeitschritte
 - ▶ Ausgabe der Temperaturmatrix in jedem Schritt

Simulationsaufbau - Setup

- Gittergrößen: 100x100, 150x150, 1500x1500
- JupyterHPC (Python 3.12.8, NumPy)
- ADIOS2 BP Engine
- Datenstruktur: temperature (2D Array float64), step (Skalar int32)
- Messung: Mittelwert aus 10 Wiederholungen

Visualisierung der Simulation



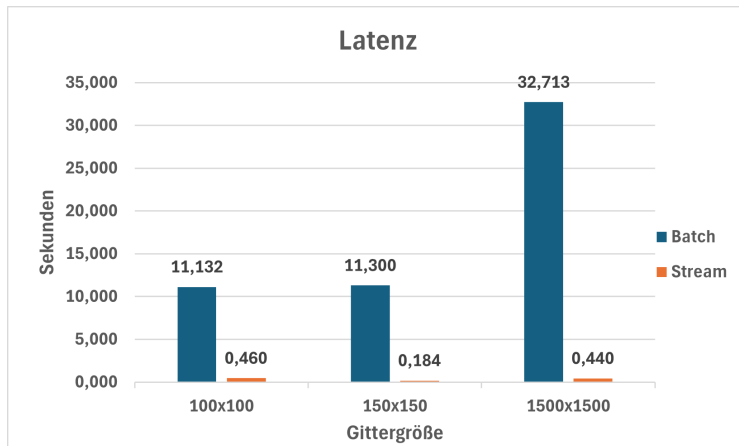
Metriken zur Bewertung

- **Latenz:** Zeit zwischen Produktion und Abschluss der ersten Analyse
 - ▶ Ziel: Minimierung (Echtzeitfähigkeit)
- **Gesamtlaufzeit:** Zeit vom Start des Prozesses bis zum Ende der Auswertung
 - ▶ **Batch:** Producer + Consumer (sequentiell)
 - ▶ **Stream:** $\text{Max}(\text{Producer}, \text{Consumer})$ (parallel)
 - ▶ Ziel: Minimierung (Effizienz)
- **Skalierbarkeit:** Verhalten bei steigender Gittergröße
 - ▶ Ziel: Geringerer Laufzeitanstieg

Agenda

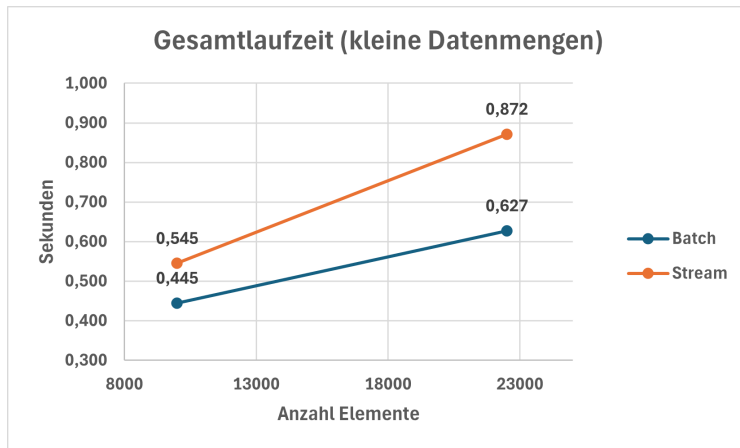
- 1 Problem
- 2 Hintergrund
- 3 Methodik
- 4 Ergebnisse**
- 5 Diskussion
- 6 Fazit

Ergebnis - Latenz



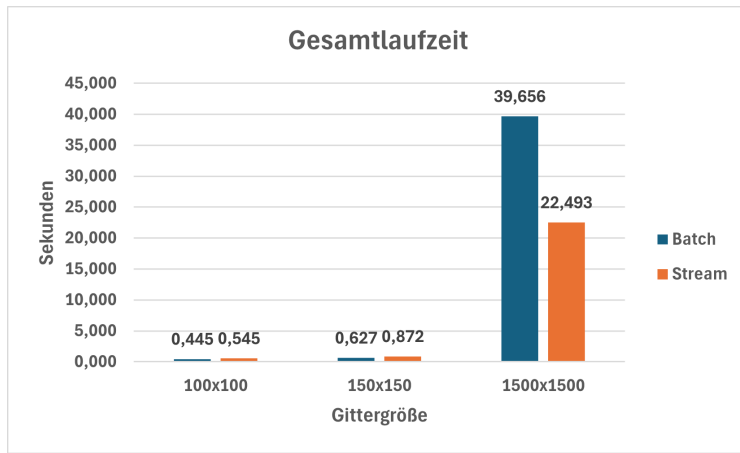
- Stream Processing ist um Faktor 25-75 schneller
- Grund: Batch wartet auf Dateiabschluss

Ergebnis - Gesamtlaufzeit (kleine Datenmengen)



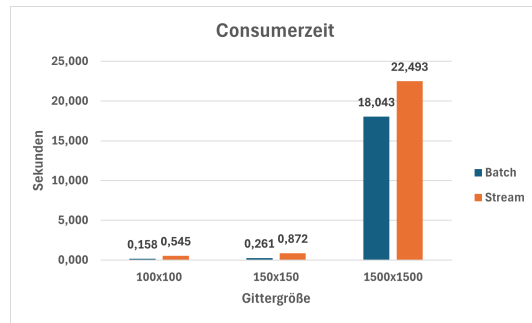
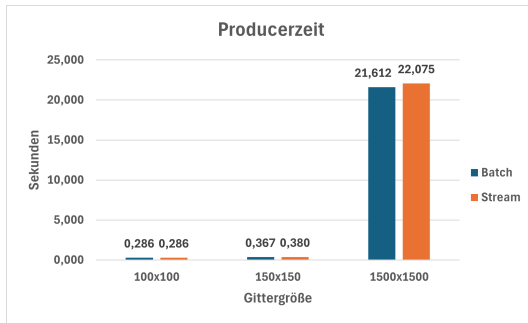
- Overhead durch Pipeline-Einrichtung beim Streaming
- Batch ist effizienter bei kleinen Datenmengen

Ergebnis - Gesamtlaufzeit



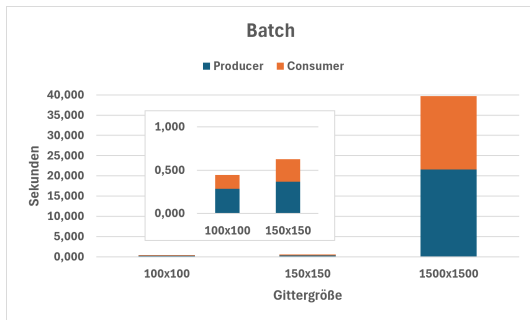
- 43 % schnellere Gesamtlaufzeit bei Stream
- Grund: Parallelität von Berechnung und Analyse

Ergebnis - Skalierbarkeit

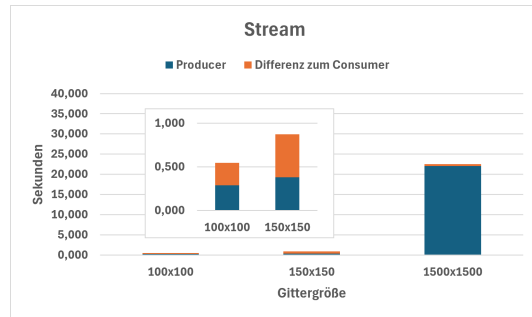


Stream macht einzelne Schritte nicht schneller, sondern überlappt sie

Ergebnis - Skalierbarkeit



Batch-Gesamtlaufzeit = Producer +
Consumer



Stream-Gesamtlaufzeit =
 $\max(\text{Producer}, \text{Consumer})$

Agenda

- 1 Problem
- 2 Hintergrund
- 3 Methodik
- 4 Ergebnisse
- 5 Diskussion**
- 6 Fazit

I/O-Entkopplung ist der Schlüssel zur Performance

Batch Processing

- Sequentiell: Consumer wartet auf Producer
- Abhängigkeit vom Dateisystem
- Leerlaufzeiten

Stream Processing

- Producer und Consumer arbeiten parallel
- ADIOS2 Buffering optimiert die Datenübertragung
- Maximale Auslastung

Für HPC-Workflows mit großen Datenmengen ist Stream überlegen

In unserem Benchmark besonders bei großer Gittergröße (I/O- und Analysezeiten fallen stärker ins Gewicht)

Wann Batch sinnvoll ist

■ Szenarien:

- ▶ sehr kleine Datensätze (Overhead dominiert)
- ▶ keine Echtzeitanforderung (Post-Processing reicht)
- ▶ einfache Reproduzierbarkeit (persistente Zwischenergebnisse erleichtern Wiederholbarkeit und Fehleranalyse)

■ Risiko Stream:

- ▶ Datenverlust bei Abbruch (keine persistente Datei)
- ▶ höhere Komplexität der Infrastruktur

Praktische Implikationen für HPC-Workflows

- **I/O-Wall:** Batch kann die I/O-Wall verstärken
- **Ressourcennutzung:** Stream nutzt Ressourcen effizienter durch Parallelität
- **Forschungseffizienz:** Niedrige Latenz ermöglicht frühere Entscheidungen
- **Zukunft:** Tendenz zu Streaming für datenintensive Workflows
- **Hybride Ansätze:** Kombination von Batch und Stream als mögliche Lösung

Agenda

- 1 Problem
- 2 Hintergrund
- 3 Methodik
- 4 Ergebnisse
- 5 Diskussion
- 6 Fazit**

Fazit

Stream Processing ist ein vielversprechender Ansatz für datenintensive HPC-Workflows mit Echtzeit- oder In-situ-Analysanforderungen.

Fazit

- **Hauptergebnis:** Latenzreduktion um bis zu 98 % durch Stream Processing
- **Effizienz:** Bei großen Datenmengen bis zu 43 % schnellere Gesamtlaufzeit
- **Empfehlungen:**
 - ▶ **Stream Processing** für Echtzeit-Analyse und große Datenmengen
 - ▶ **Batch Processing** für kleine, statische Workloads
- **Ausblick:** Hybride Ansätze, automatisierte Entscheidung

References I

- Ali, Arshad et al. "The Role of High Performance Computing Architectures". In: *Spectrum of Engineering Sciences* (Mar. 2, 2025), pp. 885–908. ISSN: 3007-3138. URL: <https://thesesjournal.com/index.php/1/article/view/158> (visited on 05/04/2026).
- Apache Flink Documentation. URL: [//nightlies.apache.org/flink/flink-docs-release-2.2/](https://nightlies.apache.org/flink/flink-docs-release-2.2/) (visited on 05/07/2026).
- Duffy, Daniel J. *Finite Difference Methods in Financial Engineering: A Partial Differential Equation Approach*. Google-Books-ID: VVjjAQAAQBAJ. John Wiley & Sons, Oct. 28, 2013. 452 pp. ISBN: 978-1-118-85648-2.
- Eisenhauer, Greg et al. "Streaming Data in HPC Workflows Using ADIOS". In: *Proceedings of the Cray User Group*. CUG 2024: Cray User Group. Perth Australia: ACM, May 5, 2024, pp. 31–43. ISBN: 979-8-4007-1328-6. DOI: 10.1145/3725789.3725793. URL: <https://dl.acm.org/doi/10.1145/3725789.3725793> (visited on 05/04/2026).
- lhde, Nina et al. "A Survey of Big Data, High Performance Computing, and Machine Learning Benchmarks". In: *Performance Evaluation and Benchmarking*. Ed. by Raghunath Nambiar and Meikel Poess. Vol. 13169. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 98–118. ISBN: 978-3-030-94436-0 978-3-030-94437-7. DOI: 10.1007/978-3-030-94437-7_7. URL: https://link.springer.com/10.1007/978-3-030-94437-7_7 (visited on 05/03/2026).

References II

Introduction — ADIOS2 2.12.1 documentation. URL:

<https://adios2.readthedocs.io/en/v2.12.1/introduction/introduction.html#what-adios2-is-and-isn-t> (visited on 05/07/2026).

JetStream | NATS Docs. Dec. 19, 2025. URL: <https://docs.nats.io/nats-concepts/jetstream> (visited on 05/07/2026).

JupyterHPC : Documentation for HPC. URL: <https://docs.hpc.gwdg.de/services/jupyterhpc/index.html> (visited on 05/11/2026).

Luckow, Andre, George Chantzalexou, and Shantenu Jha. *Pilot-Streaming: A Stream Processing Framework for High-Performance Computing.* Nov. 11, 2018. DOI: 10.48550/arXiv.1801.08648. arXiv: 1801.08648[cs]. URL: <http://arxiv.org/abs/1801.08648> (visited on 04/30/2026).

Poeschel, Franz et al. "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*. Ed. by Jeffrey Nichols et al. Vol. 1512. Series Title: Communications in Computer and Information Science. Cham: Springer International Publishing, 2022, pp. 99–118. ISBN: 978-3-030-96497-9 978-3-030-96498-6. DOI: 10.1007/978-3-030-96498-6_6. URL: https://link.springer.com/10.1007/978-3-030-96498-6_6 (visited on 05/04/2026).

Pulsar Overview | Apache Pulsar. URL: <https://pulsar.apache.org/docs/next/concepts-overview/> (visited on 05/07/2026).

References III

- Siddik, Md Saeed, Hao Li, and Cor-Paul Bezemer. "A systematic literature review of software engineering research on Jupyter notebook". In: *Journal of Systems and Software* 235 (May 2026), p. 112758. ISSN: 01641212. DOI: 10.1016/j.jss.2025.112758. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0164121225004273> (visited on 05/11/2026).
- Vinnakota, Santosh. "Combining Batch and Stream Processing for Hybrid Data Workflows". In: *International Journal on Science and Technology* 15.1 (Feb. 7, 2024), p. 2819. ISSN: 2229-7677. DOI: 10.71097/IJSAT.v15.i1.2819. URL: <https://www.ijSAT.org/research-paper.php?id=2819> (visited on 05/04/2026).

Batch Producer Code

```
#start timing
start_time = time.time()
```

```
#simulation parameters
nx, ny = 100, 100
steps = 100
alpha = 0.25
```

```
#initial grid
current = np.zeros((nx,ny), dtype=float)
current[50:60,50:60] = 100.0
```

```
#opening adios2 engine for batch writing
with ad2.Stream('heat_batch.bp','w') as s:
    for step in range(steps + 1): #Loops through all steps including the initial
        if step > 0: #skips diffusion ob step 0 -> initial condition is sent first
            new = current.copy()
            new[1:-1,1:-1] = current[1:-1,1:-1] + alpha * (current[0:-2,1:-1] + current[2:,1:-1] + current[1:-1,0:-2] + current[1:-1,2:] - 4*current[1:-1,1:-1])
            current = new

        #write Step
        s.begin_step() #starts a new ADIOS2 write ste
        s.write('temperature',current) #writes the full temperature grid under name temperature
        s.write('step', np.array(step,dtype = np.int32)) #writes the step number as a scalar array
        s.end_step() #publishes the step to the stream

    if step == 0:
        first_step_sent = time.time()

#End timing
total_runtime = time.time() - start_time
```

Batch Consumer Code

```
#start timing
start_time = time.time()

#open adios2 stream for reading
with ad2.Stream('heat_batch.bp', 'r') as s:
    step_count = 0
    for step in s.steps():
        #Read data
        temperature = s.read('temperature')
        step_num = s.read('step')

        #compute and store metrics
        max_temp = np.max(temperature)
        avg_temp = np.mean(temperature)

        max_values.append(max_temp)
        avg_values.append(avg_temp)

    #first result latency
    if step_count == 0:
        first_step_received = time.time()

    step_count += 1

#End timing
total_runtime = time.time() - start_time
```

Stream Producer Code

```
#start timing
start_time = time.time()

#simulation parameters
nx, ny = 100, 100
steps = 100
alpha = 0.25

#initial grid
current = np.zeros((nx,ny), dtype=float)
current[50:60,50:60] = 100.0

#opening a stream for writing
with ad2.Stream('heat_stream.bp', 'w') as s:
    for step in range(steps + 1): #Loops through all steps including the initial
        if step > 0: #skips diffusion ob step 0 -> initial condition is sent first
            new = current.copy()
            new[1:-1,1:-1] = current[1:-1,1:-1] + alpha * (current[0:-2,1:-1] + current[2:,1:-1] + current[1:-1,0:-2] + current[1:-1,2:] - 4*current[1:-1,1:-1])
            current = new

        #write Step
        s.begin_step() #starts a new ADIOS2 write step
        s.write('temperature', current) #writes the full temperature grid under name temperature
        s.write('step', np.array(step, dtype = np.int32)) #writes the step number as a scalar array
        s.end_step() #publishes the step to the stream

    if step == 0:
        first_step_sent = time.time()

#End timing
total_runtime = time.time() - start_time
```

Stream Consumer Code

```
#open adios2 stream for reading
start_wait = time.time()
while not os.path.exists('heat_stream.bp'):
    time.sleep(0.1)
waiting_time = time.time() - start_wait

#start timing
start_time = time.time()

with ad2.Stream('heat_stream.bp', 'r') as s:
    step_count = 0
    for step in s.steps():
        #Read data
        temperature = s.read('temperature')
        step_num = s.read('step')

        #compute and store metrics
        max_temp = np.max(temperature)
        avg_temp = np.mean(temperature)

        max_values.append(max_temp)
        avg_values.append(avg_temp)

    #first result latency
    if step_count == 0:
        first_step_received = time.time()

    step_count += 1

#End timing
total_runtime = time.time() - start_time
```