Technische Universität München
Lehrstuhl für Datenverarbeitung
Prof. Dr.-Ing. Klaus Diepold

High Performance Computing for Machine Intelligence: Gruppe 3

Autoren: *Till Hülder*, *Tobias Klama*, *Tobias Krug*

***Zusammenfassung*— Verloren in Raum und Zeit? Nicht mehr! Für alle die regelmäßig eine Ausfahrt auf dem Weg von Terra nach Alpha Centauri verpassen und unterwegs mit leerem Tank auf einem leeren Planeten landen, haben wir eine optimale Lösung entwickelt: skalierbare asynchrone Value Iteration per Open MPI. Ziel dieser Ausarbeitung ist die Einführung in die relevanten Hintergründe zu Open MPI und darauf aufbauend die Motivation eines Projektaufbaus, der die Beurteilung verschiedener Kommunikationsschemata und Parametrierungen erlaubt. Mittels dieses Frameworks können wir aus drei MPI Schemata, sechs Ausführungsumgebungen und diversen Parameterkombinationen je nach Größe des Problems und zur Verfügung stehender Rechenumgebung eine zielführende Kombination ableiten. Die Kernergebnisse sind die Identifikation verschiedener Zusammenhänge zwischen MPI Kommunikationsschema, Rechenumgebung und Parametrierung und Qualitätsmetriken wie Rechenzeit, Speicherbedarf und Lösungsqualität. Diese erlauben eine optimale Anpassung des Projekts an die jeweiligen Rahmenbedingungen.**

Keywords*—*High Performance Computing, Parallel Processing, Reinforcement Learning, Machine Learning

I. Einführung

HIGH Performance Computing bezeichnet seit einiger Zeit eine Technik zur Verknüpfung einzelner Standardcomputer zu einem leistungsfähigen Konglomerat: „In 1988, an article appeared in the Wall Street Journal titled “Attack of the Killer Micros” that described how computing systems made up of many small inexpensive processors would soon make large supercomputers obsolete.“ [1, S. 3] Entsprechend dieser Vision können wir heute auf Systeme zurückgreifen, die aus Standardcomputern performante Cluster bilden.

Diese Arbeit befasst sich mit der Implementierung eines Optimierungsproblems aus dem Reinforcement Learning Umfeld auf genau solchen Clustern. Das Problem, welches wir lösen ist die Suche einer realisierbaren und – bezogen auf eine Kostenfunktion – optimalen Route zwischen zwei Planeten in einem theoretischen Raumfahrtnavigationsszenario. Hierzu wenden wir den bekannten Value Iteration Algorithmus (TODO: Referenz?) in seiner asynchronen Form an. Die Implementierung der Value Iteration (TODO: abbreviations einbauen) erfolgte mittels C++ und dem Open MPI (TODO: ref) Framework.

Die vorliegende Ausarbeitung befasst sich mit der abstrakten Idee der Umsetzung des oben genannten Projekts und der Struktur der Testautomatisierung. Weiterhin wird eine Analyse und Einordnung der Resultate vorgenommen. Für detaillierte Einblicke in die Implementierung verweisen wir auf die Softwaredokumentation in Form der Markdown Readme Datei und Doxygen Dokumentation.

Das Hauptmerkmal unserer Ausarbeitung ist die umfangreiche Durchführung von Benchmarks mittels Variation der Größen Datensatz, Testumgebung, MPI Schema und MPI Parametrierung.

Die Umsetzung fußt auf einer konkreten Formulierung eines Projektplans, welcher Inhalt und Umfang des Projekts absteckt. Um das Ziel einer funktionsfähigen Implementierung und einer aussagekräftigen Analyse zu erreichen, setzen wir auf Ansätze der SCRUM Methodik, um mittels regelmäßiger Meetings und ausgeprägter Nutzung von Issues und Branches regelmäßigen Fortschritt zu erreichen.

Diese Ausarbeitung startet in II mit einer Erläuterung der Projektstruktur und zeigt darauf aufbauend welche Testmöglichkeiten sich hiermit bieten. Anhand dreier Schemata validieren wir die automatisierte Erfassung und Verarbeitung von Messdaten. Die so gewonnenen Ergebnisse werden in III mit einer vergleichenden Perspektive auf getestete Schemata und Ausführungsumgebungen analysiert. In IV behandeln wir konkrete Thesen, welche im HPC (TODO: abbreviation) Kontext auftreten. Den inhaltlichen Abschluss bilden eine Darstellung unserer Beiträge in V und eine Aufstellung der wesentlichen Erkenntnisse in VI. Für weitergehende Einblicke in die Ergebnisse der Arbeit, schlüsseln wir im Anhang in -A die Ergebnisse je Datensatz, Testumgebung und MPI Schema auf.

II. Methodik

Zur erfolgreichen Durchführung umfassender Benchmarks setzen wir auf eine flexible Softwarearchitektur, welche eine einfache Parametrierung von vorhandenen MPI Schemata

A. Softwarearchitektur

- Parameter structs als leichtgewichtige Umsetzung des Flyweight Patterns [2] - MVC [3], Model: VI, Controller:

MPI Schemes, View: Main + Configuration Parser + Logging <> Datenaustausch mittels Flyweight

B. Automatisierung

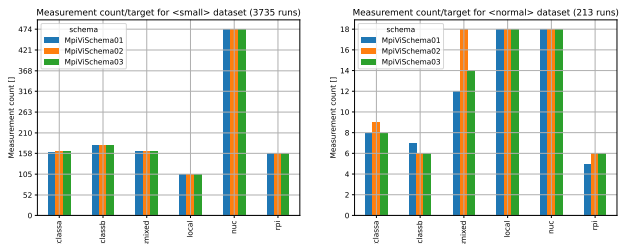
- nrun als innere Schleife war suboptimal, äußere wäre besser gewesen

C. Ausführungsumgebungen für Tests

D. Schemata

III. Analyse & Diskussion

Ziel dieses Kapitels ist es Parameter die Einfluss auf die Berechnung nehmen hervorzuheben und die drei oben erwähnten implementierten Schemen zu analysieren. Dabei soll der Fokus vorallem auf der Rechenzeit, den Speicherbedarfs und den Rechenfehler liegen.
Um die Schemta zu Vergleichen wurden Testläufe mit unterschiedlichen Parametern gemessen. Diese Ergebnisse werden in Unterkapitel A erörtert. Um erworbene Erkenntnisse auf anderen Systemen zu verifizieren wurden Messungen auf verschiedenen Klassen an Recheneinheiten ausgeführt. Dies wird in Unterkapitel B beschrieben. Zu den verwendeten Klassen gehören: HPC Klasse A (HPC 1 - HPC 5), HPC Klasse B (HPC 6 - HPC 15), eine gemischte HPC Klasse (HPC 1 - HPC15) und aus privat stammendem Besitz Respberry Pi Klasse, NUC Rechnerklasse und eine lokale Rechnerklasse.
Da es teilweise auf den Messgeräten zu einer ungleichmäßigen Auslastung kam und damit Datenausreißer generiert wurden, wurden pro Messzyklen mehrere Messungen durchgeführt. Die Anzahl und Messzeiten pro Gerät und Schmema können der Abbildung (1) entnommen werden. Auf allen Messgeräten wurden Messungen mit je dem klein und normal großen Datensatz vorgenommen. Die ins diesem Kapitel angesprochenen Grafiken und weitere Grafiken sind der Übersicht halber im Anhang abgebildet.



(a) kleiner Datensatz (b) normaler Datensatz

Abbildung 1. Anzahl an Messungen pro Rechenklasse

A. Vergleich der Schemata

Bei den Messdaten die über die Anzahl an Recheneinheiten und dem Kommunikationsintervall variieren, kann gesehen werden, dass es zwischen den einzelnen Schemen, in Bezug auf Rechenzeit und Konvergenzschritten, zu keinen großen Unterschieden kommt. Dies kann den Messungen auf den Nuc Rechnern aus der Grafik (TODO) und der Grafik(TODO) besonders gut entnommen werden. Dennoch können mit steigender Anzahl der Recheneinheiten etwas schnellere Ergebenisse erziehlt werden, siehe Grafik (TODO). Allerdings kann der Gewinn an Rechenzeit durch die Parallelisierung von Rechenschritten bei einer zu großen Anzahl an Recheneinheiten, durch den den großen Kommunikationsaufwand, schnell wieder zunichte gemacht werden, wie in Abbilung(TODO) gesehen werden kann. Die Anzahl der Recheneinheiten hat außerdem eine Auswirkung auf die Anzahl der Iterationsschritte. So steigt mit der Anzahl der Recheneinheiten auch die Anzahl der benötigten Iterationsschritte. Einen großen Einfluss auf die Rechenzeit hat das Kommunikationsintervall, siehe Grafik (nuc,run,com, small). So kann beobachtet werden, dass ganz am Anfang die Rechenzeit mit zunehmendem Kommunikationsintervall verkürzt werden kann. Doch tritt schon früh nach einer weitere Erhöhung des Kommunikationsintervalls eine Zunahme der Rechenzeit ein. Im von uns gewählten Kommunikationsintervall ist gegen Ende hin eine lineare Zunahme der Rechenzeit zu sehen, Grafik(mixed). Diese Zunahme der Rechenzeit resultiert vor allem aus einer höheren Anzahl an benötigten Iterationsschritten bis zur Konvergenz, siehe Grafik (nuc,step.small). Es wird außerdem aus der Grafik (nuc,step.small) sichtbar, dass mit einem höherem Kommunikationsintervall eine höhere Varianz bei den Iterationsschritten entsteht. Diese entstehende Varianz ist bei allen gemessenen Schemen gleich ausgeprägt.
Auch bei der Frage des Speicherbedarfs können einige Erkenntnisse gewonnen werden. Generell ist zu sehen, dass Schema 1 und Schema 3 beim Speicherbedarf nahe beieinander liegen. Schema 2 benötigt auf der Recheneinheit mit dem Rang 0 einen deutlich höheren Speicherbedarf als die anderen beiden Schemata. Wenn man jedoch den gesamten Speicher für die Recheneinheiten über die Anzahl von Recheneinheiten anschaut, wie in Grafik (classa,snall,rsssum), so sieht man dass mit höherer Anzahl an Recheneinheiten der Speicherbedarf steigt. Bei Schema 2 jedoch nicht so stark wie bei den anderen Schemata. Daher ist etwa ab 4 Recheneinheiten besser das Speicherärmer Schema 2 zu verwenden. Das könnte mit dem Schemaaufbau erklärt werden, da hier nur ein Rang alle Daten einliest und erst danach auf die anderen Rechner weiterveteilt.
Bei der Analyse des Rechenfehlers ist es schwieriger anhand der gewonnenen Messdaten eine Aussage zu treffen, da die Messergebnisse je nach Rechnerklasse variieren können. Jedoch lässt sich sagen, dass der Mittelwert bei gleicher Parameterwahl und gleicher Rechnerklasse zwischen den Schemen wenig variiert. Dies gilt sowohl für die l2, die Maxnorm und die mittlere quadratische Abweichung. Außerdem bleibt der Fehler je nach Recheneinheit mit varrierender Rechenanzahl und Kommunikationsintervall gleich, siehe Grafik (jdiff ws,small) oder Grafik (max,com,nux).

B. Vergleich der Ausführungsumgebungen

Beim Vergleich der verschiedenen Ausführungsrechenklassen fällt vorallem auf, dass die Rechenzeit auf den Nuc,

Lokalen und Raspberry PI Rechnern zwischen den implementierten Schemen weniger variiert. Da die Auslastung auf den HPC Rechnern, je nach Anzahl der Benutzer stark variiert, wird hier auch eine Varianz in den Rechenzeiten sichtbar. Da die Rechnergruppen jedoch unterschiedliche Rechenleistungen aufweisen, kann man keinen direkten Vergleich der Rechenzeit vornehmen. Dennoch können bei der Analyse der Rechenzeit auf den verschiedenen Messgerätklassen, Eigenschaften der verschiedenen Schemata aufgezeigt werden. So sieht man dass der Mittelwert der Rechenzeit bei größeren Kommunikationsintervallen in der Mixed Klasse größer ist als in Klasse B. Die Mixed Rechnerklasse HPC Rechner beinhaltet Rechner aus Klasse A und Klasse B. Dabei weist die Rechnerklasse A eine leicht schlechtere Rechenleistung auf, wie der Vergleich der mittleren Laufzeiten von Klasse A und Klasse B sich zeigt. Da nun in den implementierten Schemen bei der Kommunikation auf das langsamste Glied gewartet werden muss, kann die leicht homogen perfomantere Rechnerklasse schneller zu einem Ergebniss kommen.
Auch bei der Betrachtung des Rechenfehlers gab es Unterschiede zwischen den Rechnerklassen. So die wird Berechnungen auf Rechnerklasse A mit einem größer Fehler ausgeführt als auf Rechnerklasse B.
Beim Vergleich der unterschiedlichen Ausführungsergebnissen konnte jedoch meistens die Erkenntnisse aus dem Unterkapitel A auf allen Rechnerklassen bestätigt werden.

IV. Thesen

A. Es besteht eine Korrelation RAM mit world_size, nach einer Kurzgeschichte von Hans Mueller

blabla, siehe Figure 3 bis 17

B. Es besteht eine Korrelation runtime mit com_interval

blabla

C. Es besteht eine inverse Korrelation zwischen world_size und runtime

blabla

V. Beiträge

- Testumgebung für automatisierte Analyse von Open MPI Kommunikationsschemata für asynchrone Value Iteration auf verschiedenen Ausführungsumgebungen

1) Till Hülder: III
2) Tobias Klama: IV
3) Tobias Krug: Zusammenfassung, I, II

VI. Erkenntnisse

wir konnten zeigen, dass: - automatisiertes ist tauglich/realisierbar - der Einfluss von Targets und Parametern auf die Performance von Open MPI für ein VI Problem konnte gezeigt werden

Literatur

[1] Dowd, K.: *High performance computing*, O'Reilly & Associates, Cambridge Sebastopol, CA, 1998. – ISBN 9781565923126

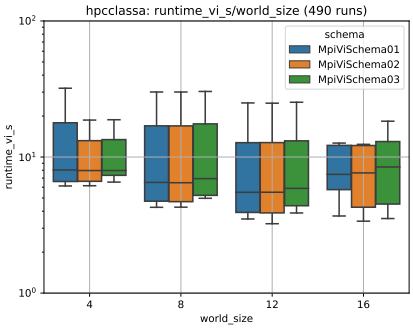
[2] Gamma, E.: *Design patterns : elements of reusable object-oriented software*, Addison-Wesley, Reading, Mass, 1995. – ISBN 9780201633610

[3] Buschmann, F.: *Pattern-oriented software architecture : a system of patterns*, Wiley, Chichester New York, 1996. – ISBN 9780471958697

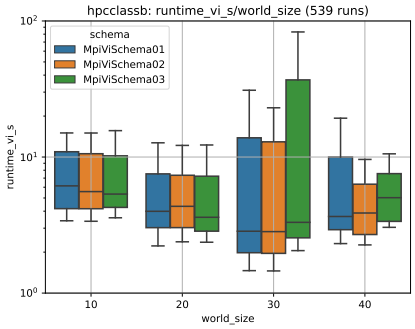
A. Benchmark

1) Benchmak Datensatz small: Die nachfolgenden Graphiken zeigen die Ergebnisse der Benchmarks für den Datensatz small.

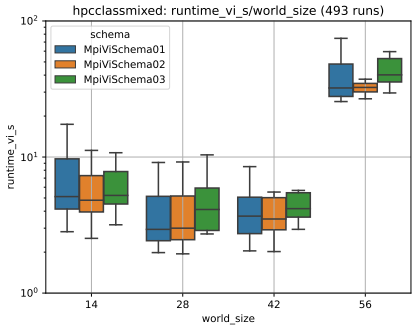
2) Benchmark Datensatz normal: Die nachfolgenden Graphiken zeigen die Ergebnisse der Benchmarks für den Datensatz normal.



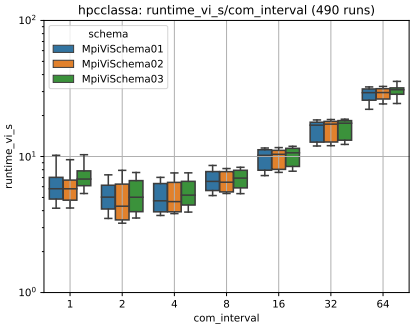
(a) HPC class A, runtime vs. world_size



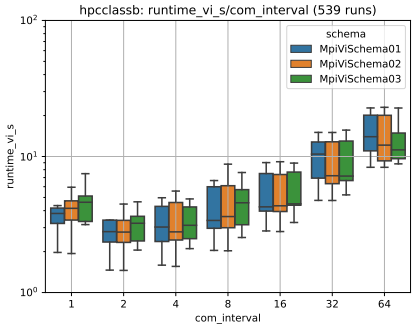
(b) HPC class B, runtime vs. world_size



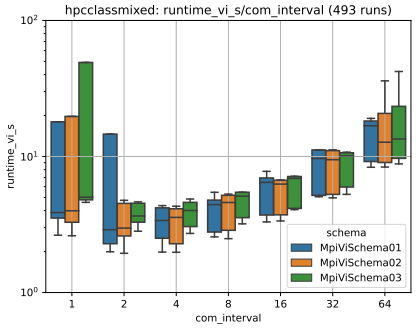
(c) HPC class mixed, runtime vs. world_size



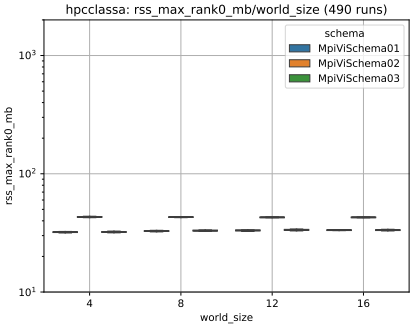
(d) HPC class A runtime vs. com_interval



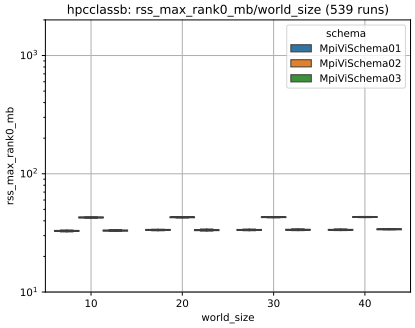
(e) HPC class B runtime vs. com_interval



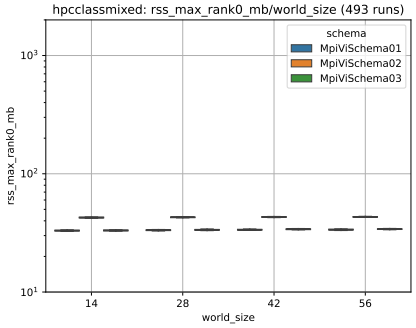
(f) HPC class mixed runtime vs. com_interval



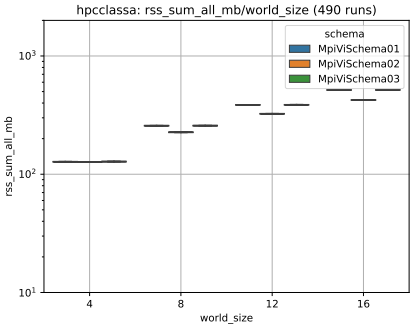
(g) HPC class A max rss rank_0 vs. world_size



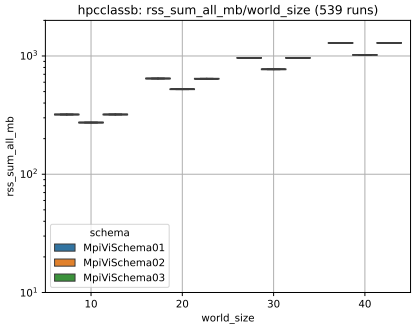
(h) HPC class B max rss rank_0 vs. world_size



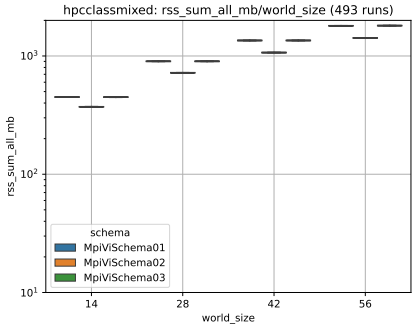
(i) HPC class mixed max rss rank_0 vs. world_size



(j) HPC class A rss-sum vs. world_size

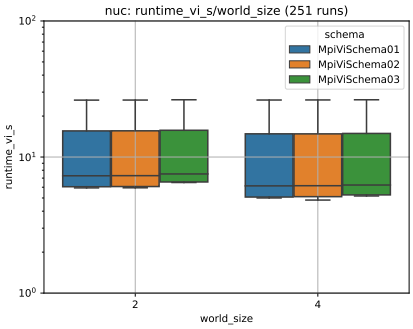


(k) HPC class B rss-sum vs. world_size

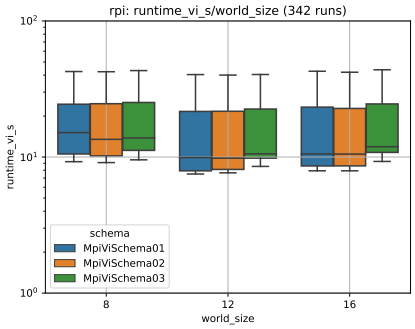


(l) HPC class mixed rss-sum vs. world_size

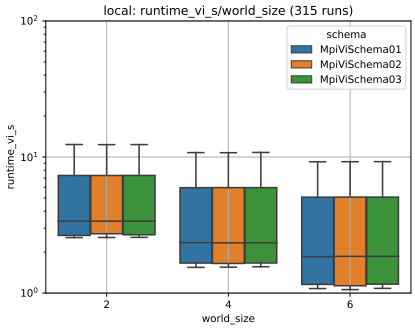
Abbildung 2. Comparison between HPC classes with dataset small



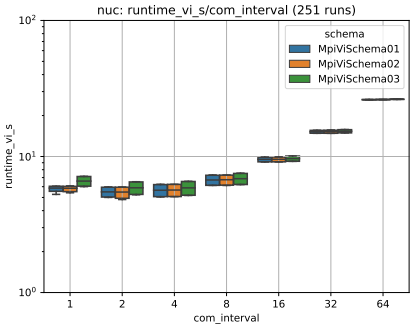
(a) NUC, runtime vs. world_size



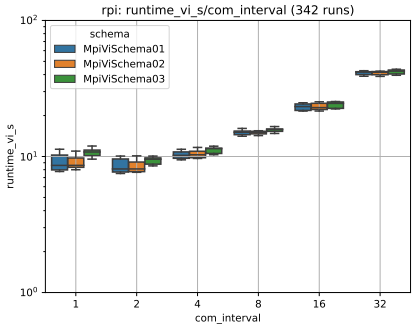
(b) RPi, runtime vs. world_size



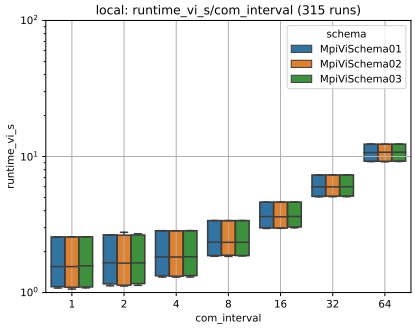
(c) Local, runtime vs. world_size



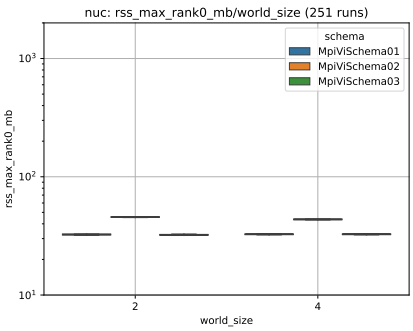
(d) NUC runtime vs. com_interval



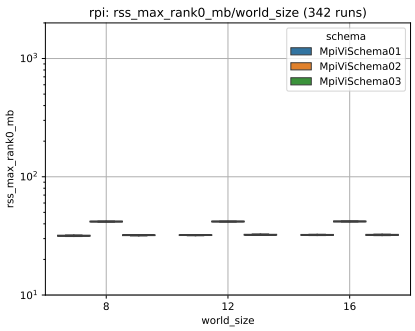
(e) RPi runtime vs. com_interval



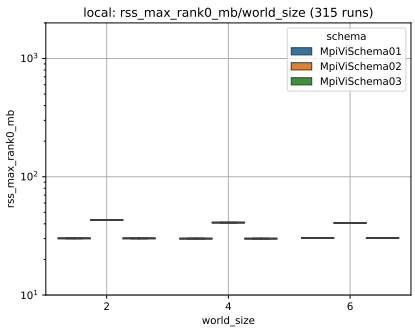
(f) Local runtime vs. com_interval



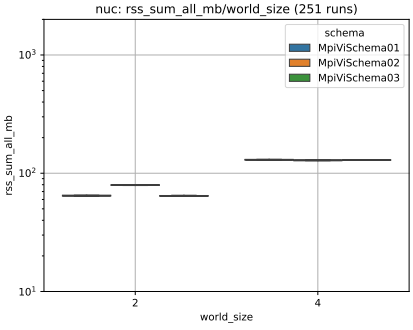
(g) NUC max rss rank_0 vs. world_size



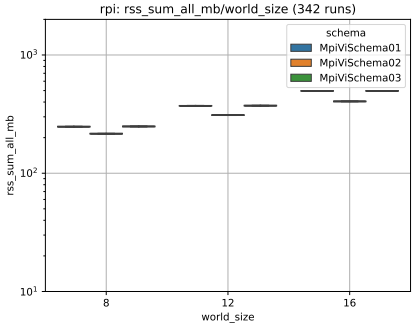
(h) RPi max rss rank_0 vs. world_size



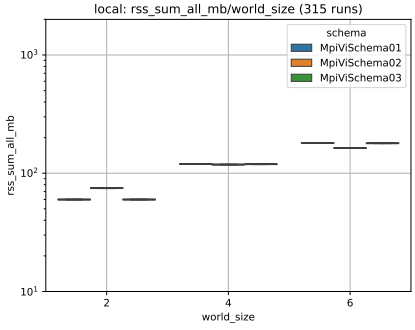
(i) Local max rss rank_0 vs. world_size



(j) NUC rss-sum vs. world_size

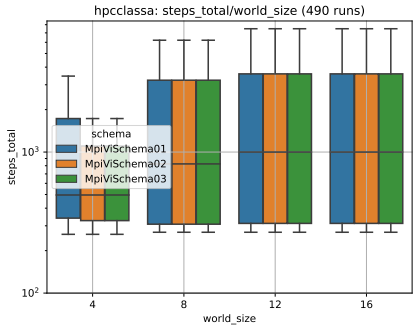


(k) RPi rss-sum vs. world_size

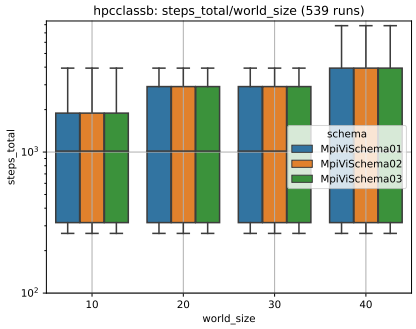


(l) Local rss-sum vs. world_size

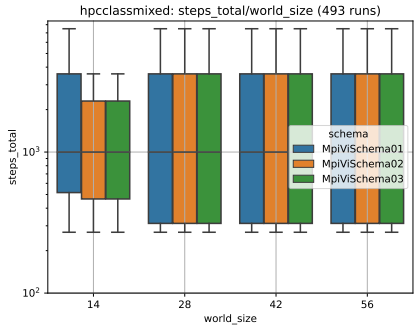
Abbildung 3. Comparison between NUC, RPi and Local with dataset small



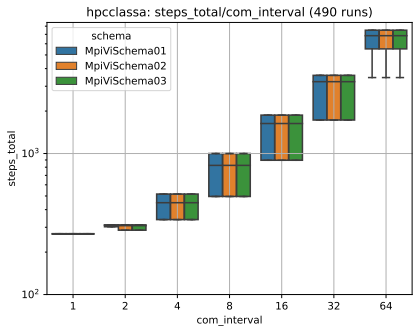
(a) HPC class A, Iterations vs. world_size



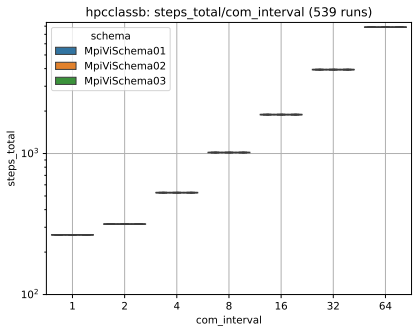
(b) HPC class B, Iterations vs. world_size



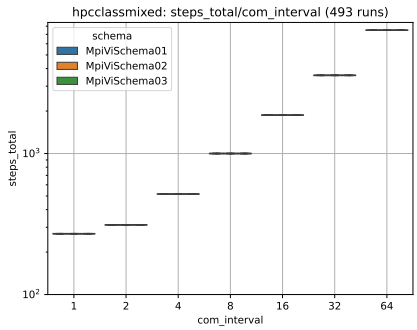
(c) HPC class mixed, Iterations vs. world_size



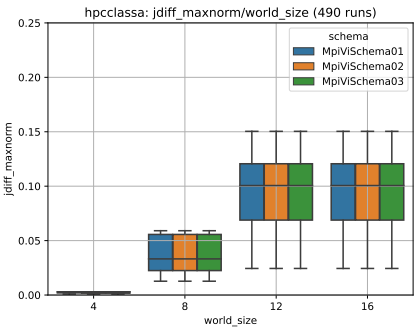
(d) HPC class A Iterations vs. com_interval



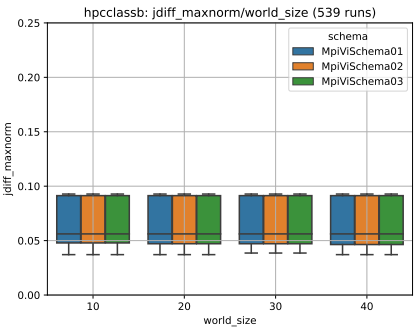
(e) HPC class B Iterations vs. com_interval



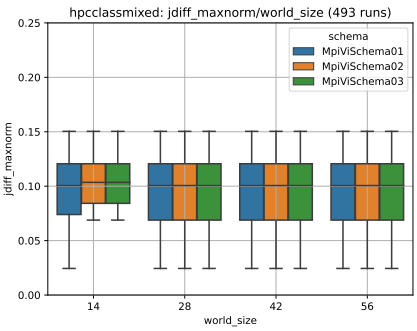
(f) HPC class mixed Iterations vs. com_interval



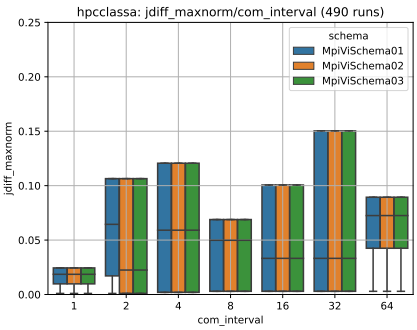
(g) HPC class A J-diff maxnorm vs. world_size



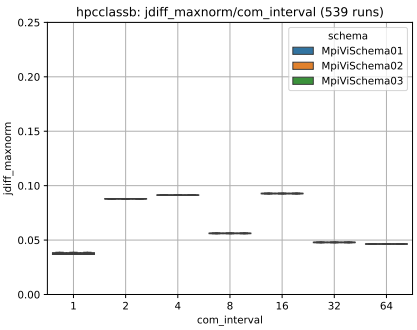
(h) HPC class B J-diff maxnorm vs. world_size



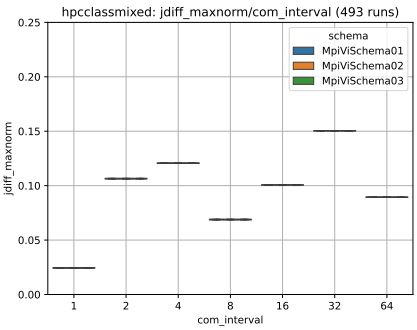
(i) HPC class mixed J-diff maxnorm vs. world_size



(j) HPC class A J-diff maxnorm vs. com_interval

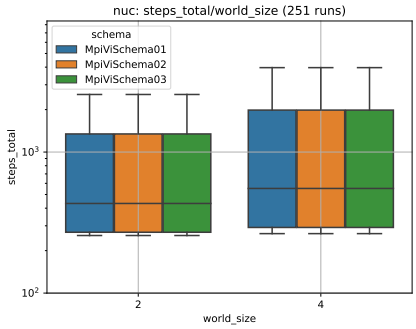


(k) HPC class B J-diff maxnorm vs. com_interval

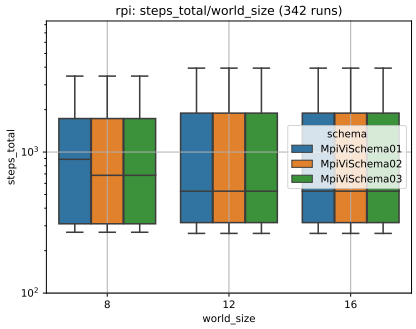


(l) HPC class mixed J-diff maxnorm vs. com_interval

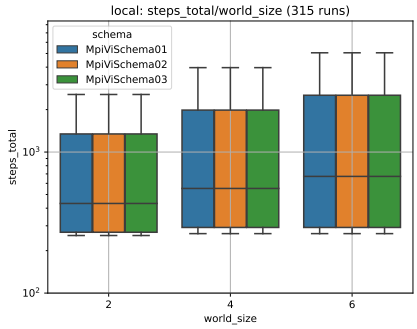
Abbildung 4. Comparison between HPC classes with dataset small



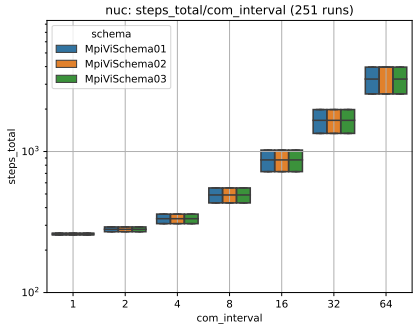
(a) NUC, Iterations vs. world_size



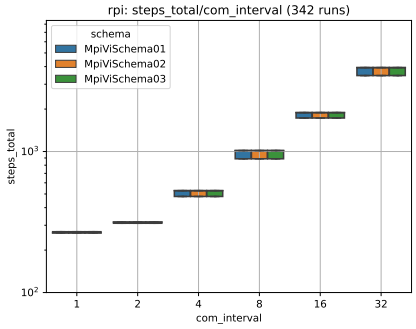
(b) RPi, Iterations vs. world_size



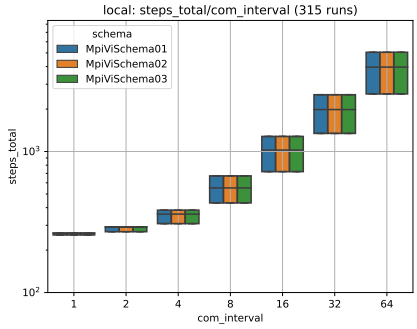
(c) Local, Iterations vs. world_size



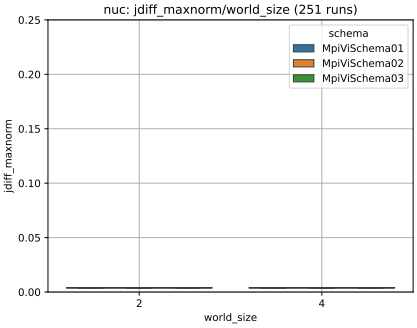
(d) NUC Iterations vs. com_interval



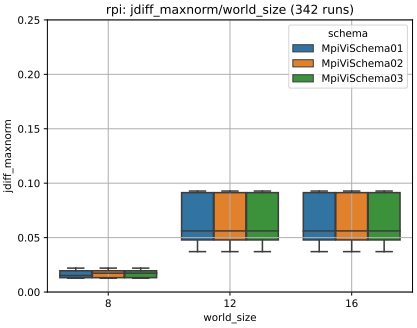
(e) RPi Iterations vs. com_interval



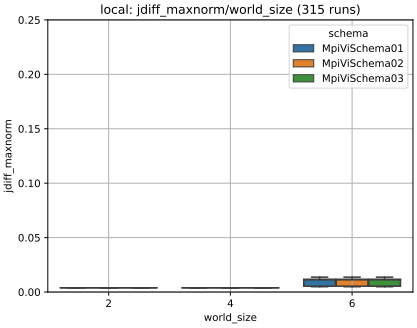
(f) Local Iterations vs. com_interval



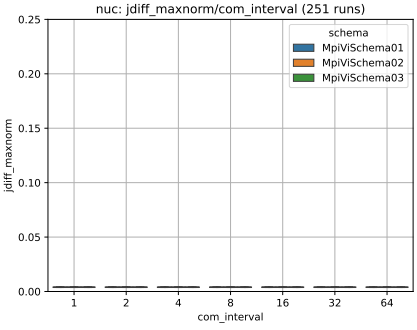
(g) NUC J-diff maxnorm vs. world_size



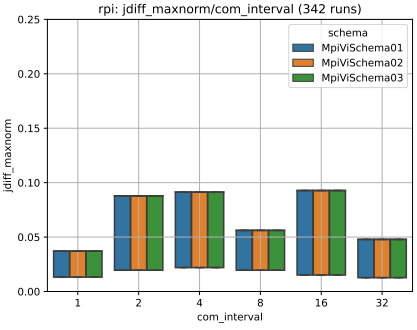
(h) RPi J-diff maxnorm vs. world_size



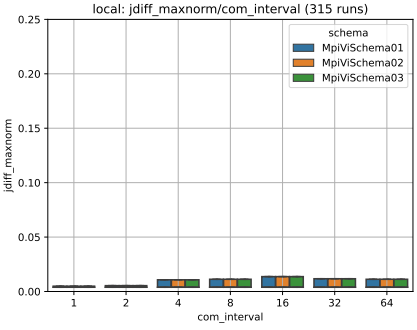
(i) Local J-diff maxnorm vs. world_size



(j) NUC J-diff maxnorm vs. com_interval

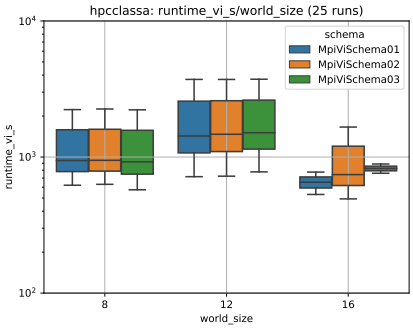


(k) RPi J-diff maxnorm vs. com_interval

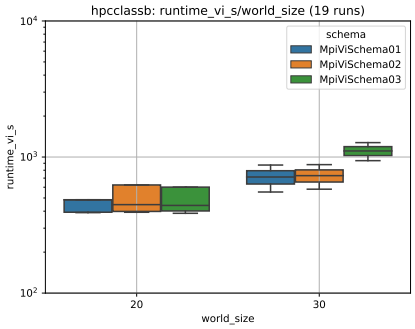


(l) Local J-diff maxnorm vs. com_interval

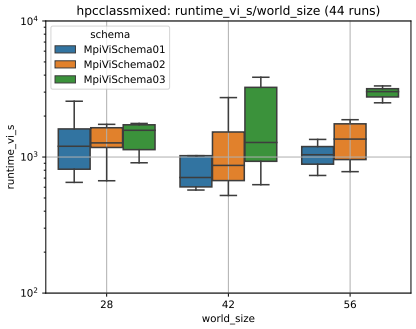
Abbildung 5. Comparison between NUC, RPi and Local with dataset small



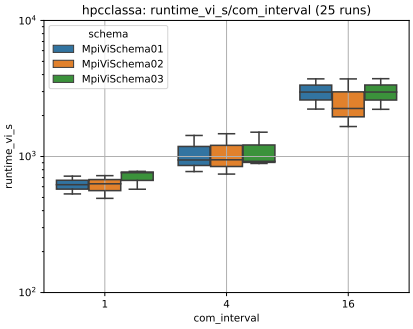
(a) HPC class A, runtime vs. world_size



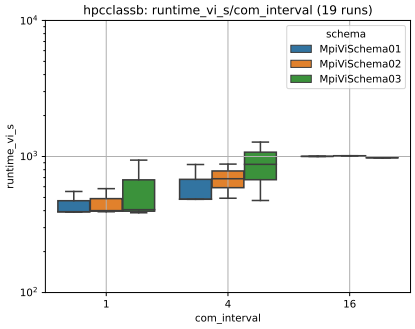
(b) HPC class B, runtime vs. world_size



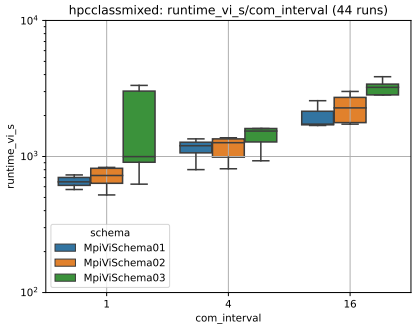
(c) HPC class mixed, runtime vs. world_size



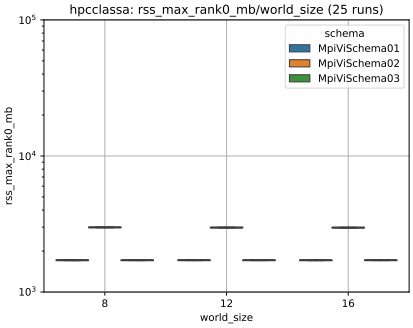
(d) HPC class A runtime vs. com_interval



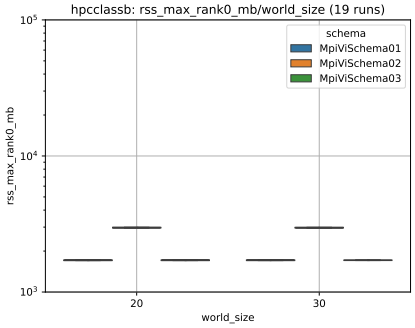
(e) HPC class B runtime vs. com_interval



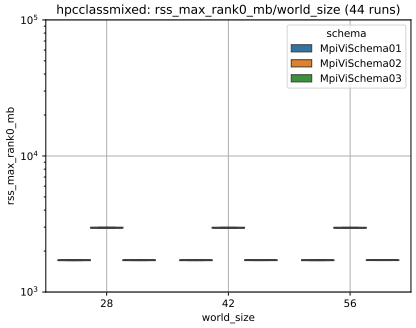
(f) HPC class mixed runtime vs. com_interval



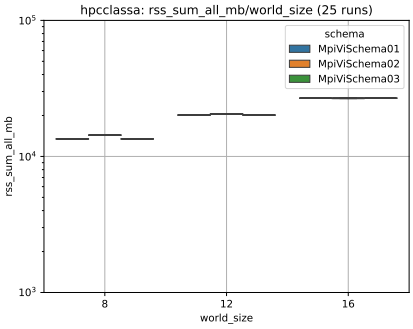
(g) HPC class A max rss rank_0 vs. world_size



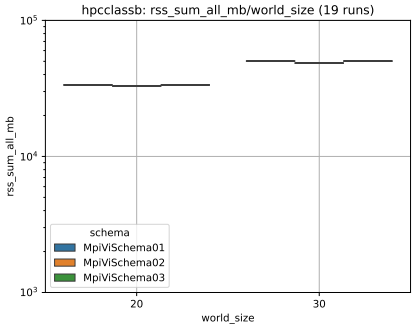
(h) HPC class B max rss rank_0 vs. world_size



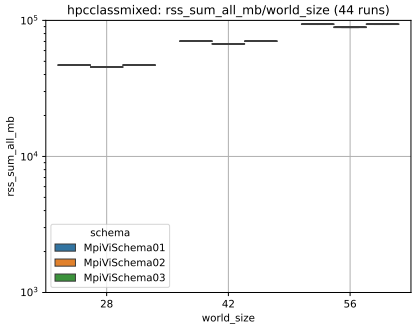
(i) HPC class mixed max rss rank_0 vs. world_size



(j) HPC class A rss-sum vs. world_size

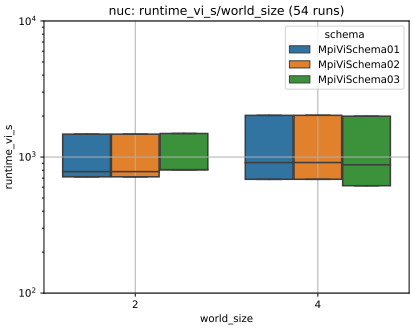


(k) HPC class B rss-sum vs. world_size

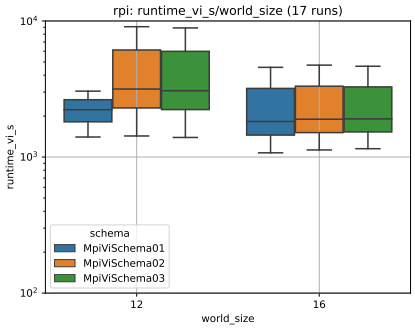


(l) HPC class mixed rss-sum vs. world_size

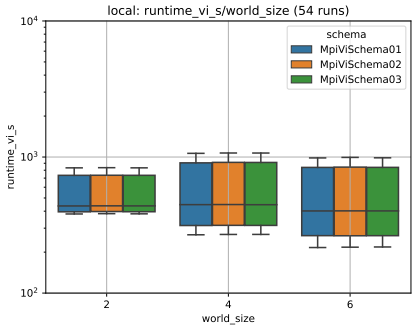
Abbildung 6. Comparison between HPC classes with dataset normal



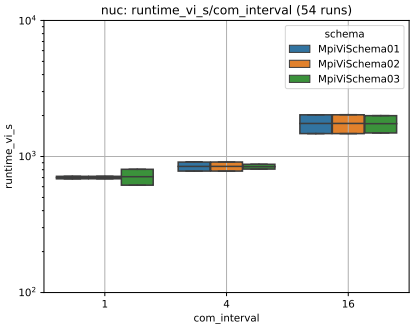
(a) NUC, runtime vs. world_size



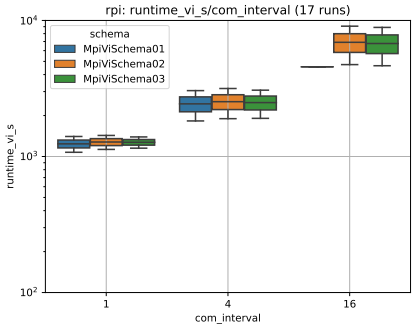
(b) RPi, runtime vs. world_size



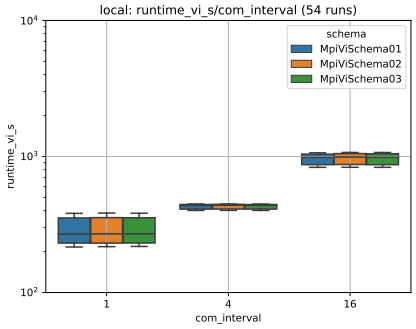
(c) Local, runtime vs. world_size



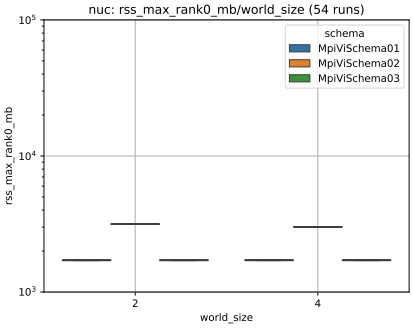
(d) NUC runtime vs. com_interval



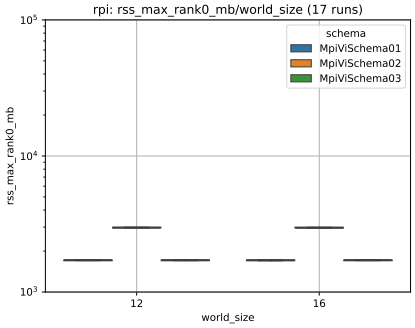
(e) RPi runtime vs. com_interval



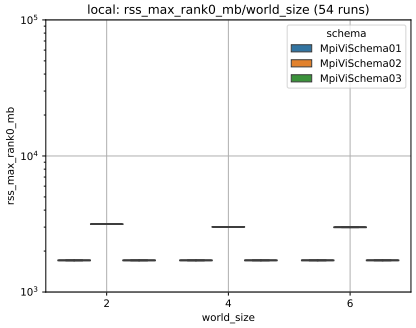
(f) Local runtime vs. com_interval



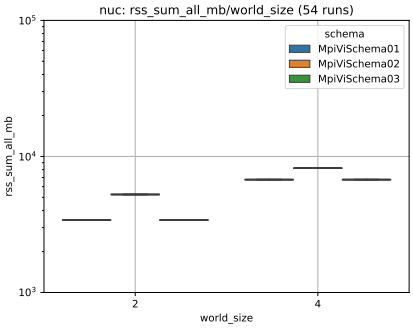
(g) NUC max rss rank_0 vs. world_size



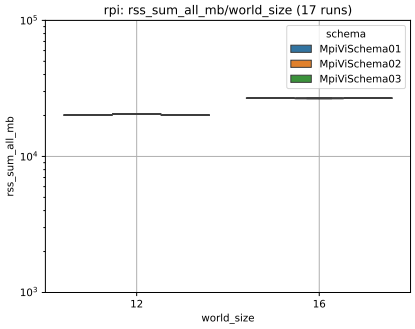
(h) RPi max rss rank_0 vs. world_size



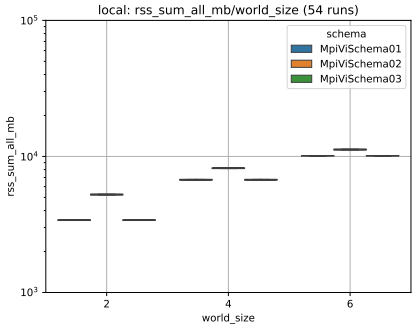
(i) Local max rss rank_0 vs. world_size



(j) NUC rss-sum vs. world_size

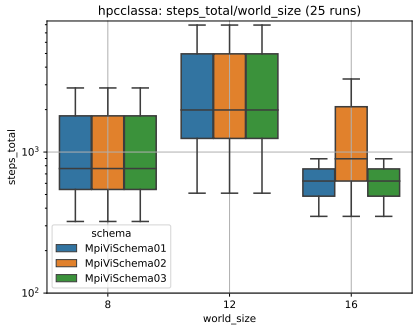


(k) RPi rss-sum vs. world_size

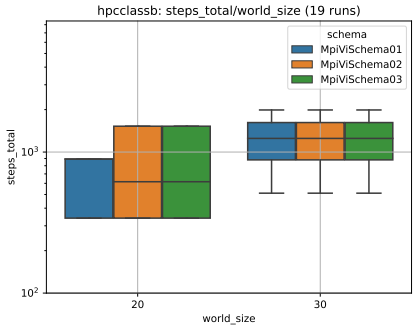


(l) Local rss-sum vs. world_size

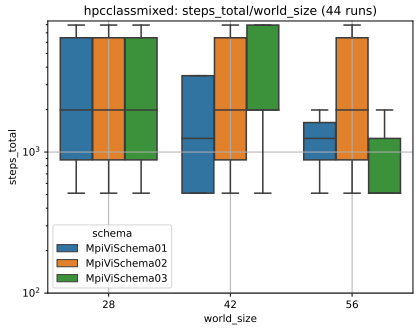
Abbildung 7. Comparison between NUC, RPi and Local with dataset normal



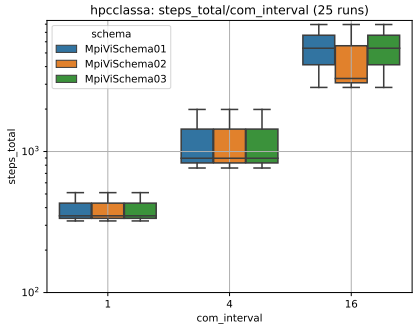
(a) HPC class A, Iterations vs. world_size



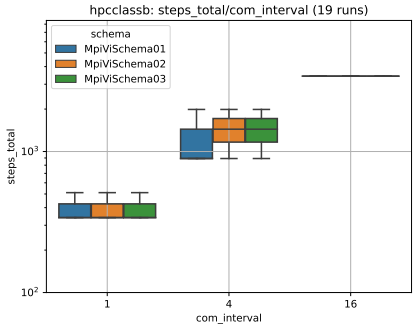
(b) HPC class B, Iterations vs. world_size



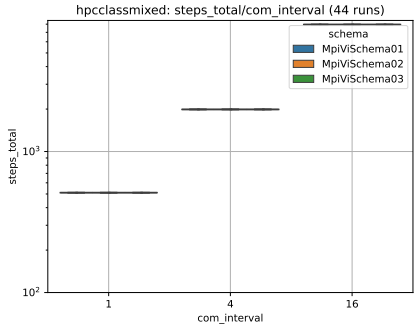
(c) HPC class mixed, Iterations vs. world_size



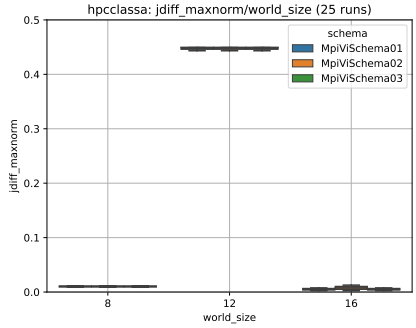
(d) HPC class A Iterations vs. com_interval



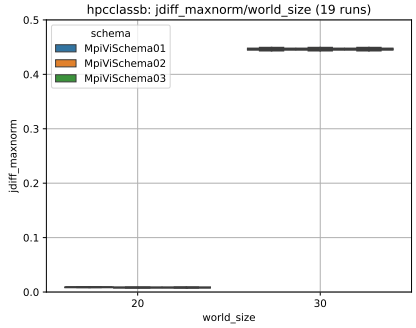
(e) HPC class B Iterations vs. com_interval



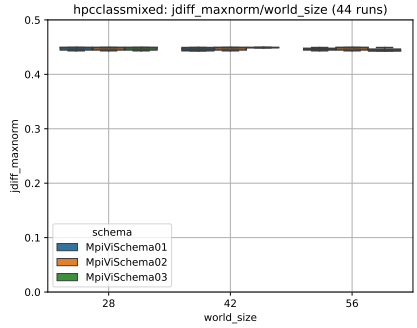
(f) HPC class mixed Iterations vs. com_interval



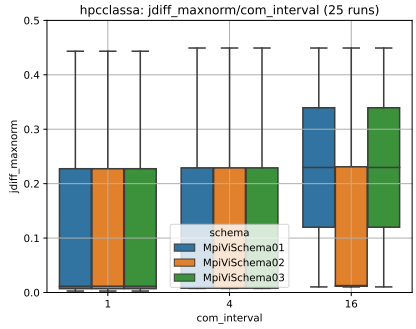
(g) HPC class A J-diff maxnorm vs. world_size



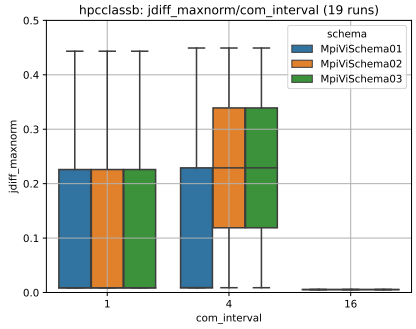
(h) HPC class B J-diff maxnorm vs. world_size



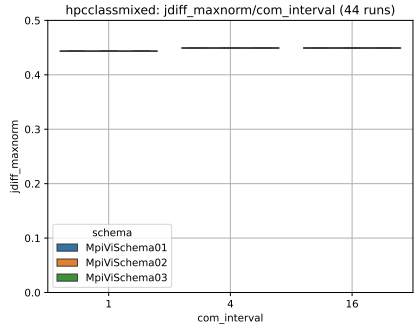
(i) HPC class mixed J-diff maxnorm vs. world_size



(j) HPC class A J-diff maxnorm vs. com_interval

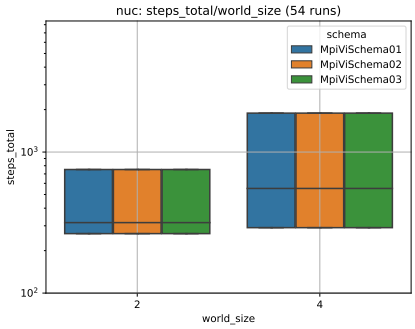


(k) HPC class B J-diff maxnorm vs. com_interval

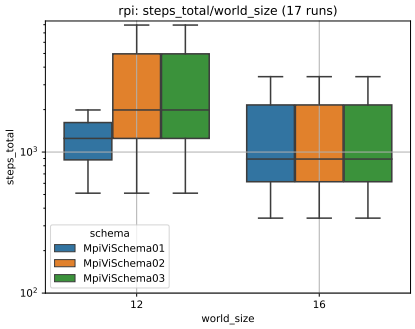


(l) HPC class mixed J-diff maxnorm vs. com_interval

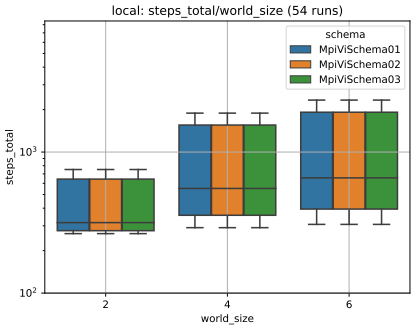
Abbildung 8. Comparison between HPC classes with dataset normal



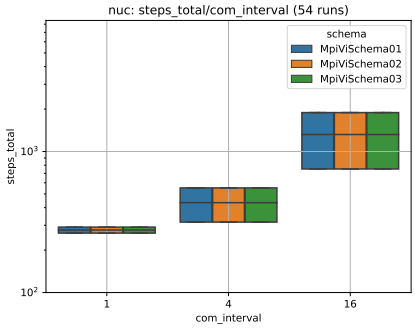
(a) NUC, Iterations vs. world_size



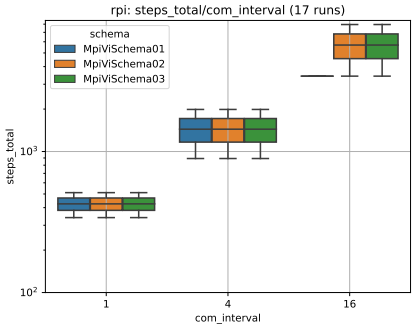
(b) RPi, Iterations vs. world_size



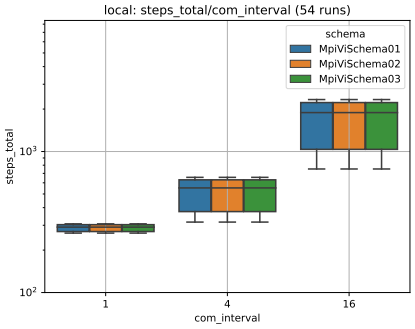
(c) Local, Iterations vs. world_size



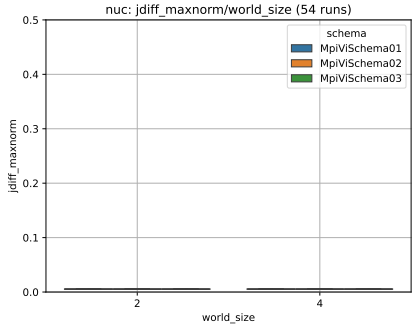
(d) NUC Iterations vs. com_interval



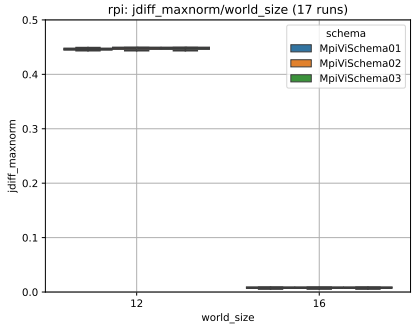
(e) RPi Iterations vs. com_interval



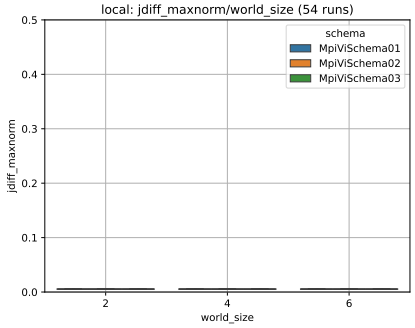
(f) Local Iterations vs. com_interval



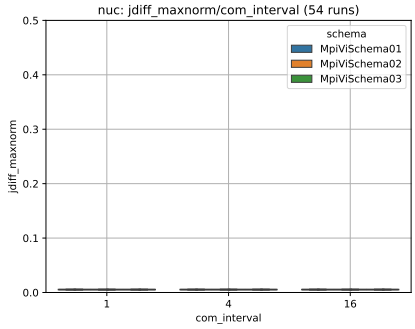
(g) NUC J-diff maxnorm vs. world_size



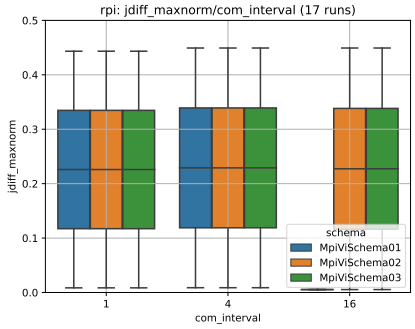
(h) RPi J-diff maxnorm vs. world_size



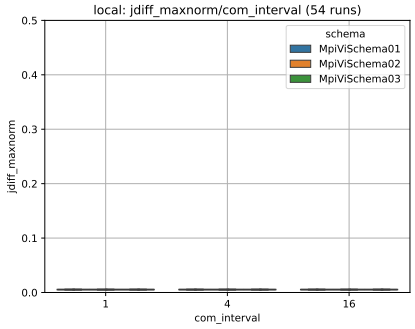
(i) Local J-diff maxnorm vs. world_size



(j) NUC J-diff maxnorm vs. com_interval



(k) RPi J-diff maxnorm vs. com_interval



(l) Local J-diff maxnorm vs. com_interval

Abbildung 9. Comparison between NUC, RPi and Local with dataset normal