

Intro HPC: Blatt 2

04.11.1014

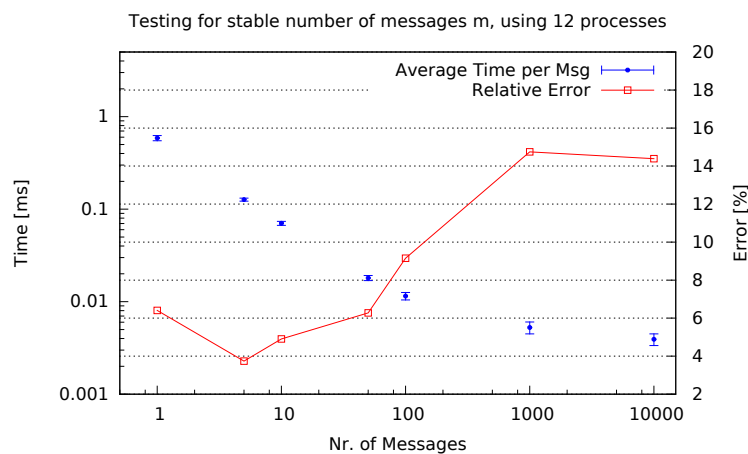
2.1 MPI Ring Communication

Der Quelltext zur aufgabe liegt unter `../2/2_1/2_1.cpp`. In dem Ordner liegt auch ein `Makefile` zum Compilieren und Ausführen. Benutzung:

<code>make</code>	Compile
<code>make clean</code>	Bin und *.o löschen
<code>make run proc=\$p msg=\$m v=\$v</code>	Ausführen mit \$p Prozessen, \$m Nachrichten
<code>make run_opt proc=\$p msg=\$m v=\$v</code>	Siehe run, aber mit optimiertem Mapping

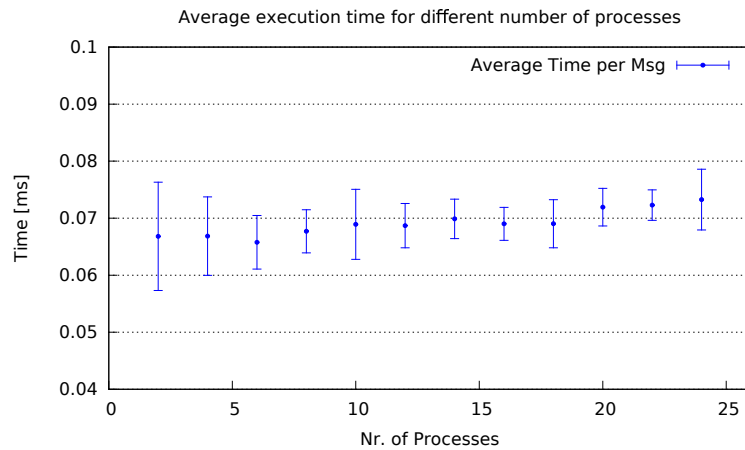
Parameter `$v` für Ausgabe der Zeit, 1 für nett formatiert, 0 nur gesamte Zeit und pro Nachricht.

Plot unten zeigt durchschnittliche Zeit pro Nachricht bei 12 Prozessen. Jeder run wurde 20 mal gemessen um den Fehler in der Laufzeit zu bestimmen (stabilste message size `m`). Obwohl 5 Nachrichten eine kleine Abweichung haben, haben wir für die Messung 10 Nachrichten genommen um ein wenig bessere Statistik machen zu können.



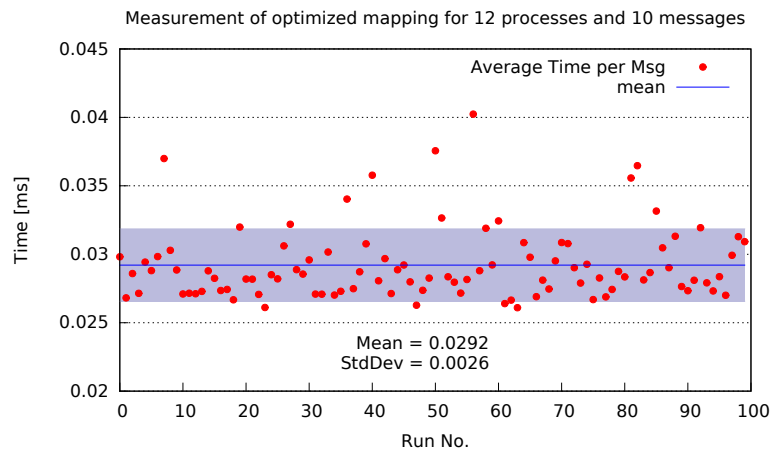
Messung der Ausführungsdauer bei 24 Prozessen, 10 Nachrichten pro Prozess und Mapping `creek01, ..., creek08`. Das Programm wurde auch wieder für jede Prozessanzahl 20 mal ausgeführt. Die Ergebnisse sind unten geplottet. Die Zeit pro Nachricht steigt mit der Anzahl Prozesse leicht an und liegt bei ≈ 0.07 ms.

Für den letzten Teil der Aufgabe wurde das Mapping für bessere Performance geändert. Jeder Prozess sendet an seinen Nachfolger, da dieser durch das "normale" Mapping



jeweils auf einem anderen Rechner liegt, muss jede Nachricht einmal durchs Netzwerk, was Zeit kostet. Das Mapping wurde so angepasst, dass die ersten 8 Prozesse (**creek04** hat laut **nproc** 8 Cores) auf **creek04** laufen und die nächsten 4 auf **creek05**. Dadurch sind nur zwei Nachrichten über das Netzwerk nötig, alle anderen laufen nur von Core zu Core in der selben Maschine.

Dadurch verringert sich die Zeit pro Nachricht auf 0.0292 ± 0.0026 ms (siehe Plot). Das optimierte Mapping ist damit ≈ 2.35 fach schneller.



2.2 Barrier Synchronization

2.3 Matrix multiply – sequential version