Alexander Schnapp
Max Menges
introhpc02

Intro HPC: Blatt 5

24.11.1014

5.1 Heat Relaxation – Sequential Implementation

Der Quelltext liegt unter `../5/heat.cpp`, da liegt auch ein `Makefile`. Zum compilieren `make` eingeben. Das copiliert dann mit Optimierung `-O2`. Zum ausführen `make run n=128 it=100` eingeben, optional zum visualisieren: `make gif n=128 it=100`. Das erzeugt ein `animation.gif` im `out` Ordner.

5.2 Heat Relaxation - Experiments

Für die Anzahl *Flops* gilt in Abhängigkeit von der Grid size n:

$$Flops = 7 \cdot n^2 - 4n + 4$$

Die Messungen wurden mit jeweils 100 Iterationen durchgeführt. Initialisierung und Output wurde nicht mitgemessen, dafür allerdings die Zeit die Benötigt wird, das Grid zu kopieren.

Man sieht, dass die GFLOPs zunächst mit steigender Grid größe ebenfalls zunehmen, ab 512x512 aber wieder abfallen.

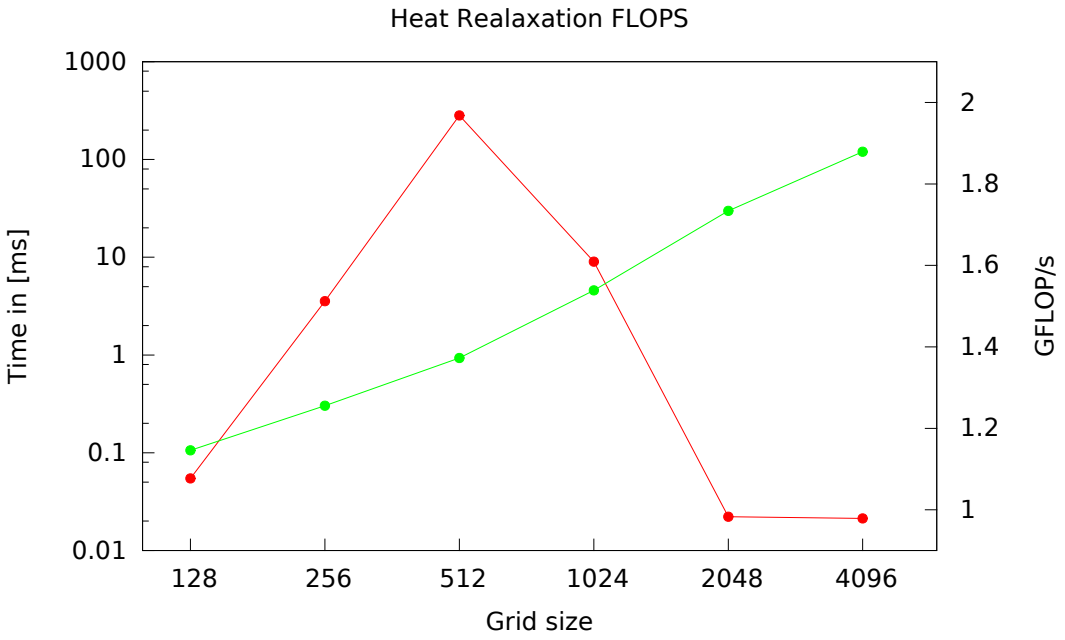
Das Problem ist zunächst Computationally bound, größere Grids führen zu mehr Flops. Abeiner Grid größe von ca. 512x512 dann aber Memory bound, da die Grids nicht mehr in den Cache passen. Bei 1024x1024 ist ein double Grid $8MB$ was fast der L3-Cache größe von $10MB$ entspricht. Es müssen aber 2 Grids (für Iterationsschritte t und $t+1$) im Chache gehalten werden.

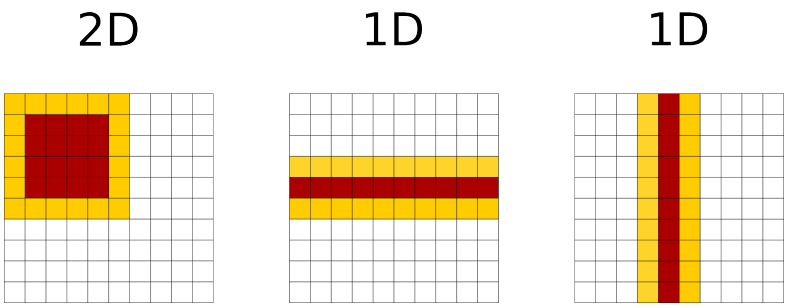
Grid size	Time/iteration	Flops total	GFLOP/s
128 x 128	0.000106	114,688	1.077
256 x 256	0.000303	458,752	1.512
512 x 512	0.000932	1,835,008	1.968
1024 x 1024	0.004582	7,340,032	1.609
2048 x 2048	0.029858	29,360,128	0.983
4096 x 4096	0.119876	117,440,512	0.979

5.3 Heat Relaxation – Pre-considerations for parallelization

Ich würde für das Problem eine 1D-Zeilenweise Zerlegung wählen. Somit wird immer zusammenhängender Speicher adressiert. Die Aufteilung geschieht wie folgt:

- Jeder Prozess P berechnet die p-te Zeile des neuen Grids X^{t+1}


Heat Realaxation FLOPS
Time in [ms]
GFLOP/s
Grid size
1000
100
10
1
0.1
0.01
2
1.8
1.6
1.4
1.2
1
128
256
512
1024
2048
4096


2D
1D
1D

- Zu Anfang bekommt jeder Prozess die Daten aus Zeile $p+1$. Also alle *unteren* Nachbarn der Elemente aus der zu berechnenden Zeile p.
- Diese können für das jeweilige Element aufsummiert werden.
- Im nächsten schritt werden die Daten von X^t an den nächsten Prozess weitegeschickt und vom vorherigen empfangen. Die neuen Daten wieder in der entprechenden Spalte aufsummieren und weiterschicken etc.
- Man bekommt (bei Aufteilung von n Zeilen auf n Prozesse) eine 3 Stufige art Pipeline. Die auch für $p < n$ Skalierbar ist.

Nachdem der letzte Datensatz verarbeitet wurde, müssen die neuen X^{t+1} Daten wieder an einem Prozess eingesamelt werden. Dies kann auch gleich zum Synchronisieren genutzt werden, wenn alle Daten von allen Prozessen vorliegen, kann die nächste Iteration gestartet werden.

Dadurch das nciht alle Daten aufeinmal versendet werden sondern (in dem Beispiel) jeweils nur ein drittel, kann man den communication overlap zum Berechnen Nutzen. Dass der mächste Schritt der Berechnung gestartet werden kann hängt nun davon ab, dass die Daten vom Vorgängerprozess vorliegen.