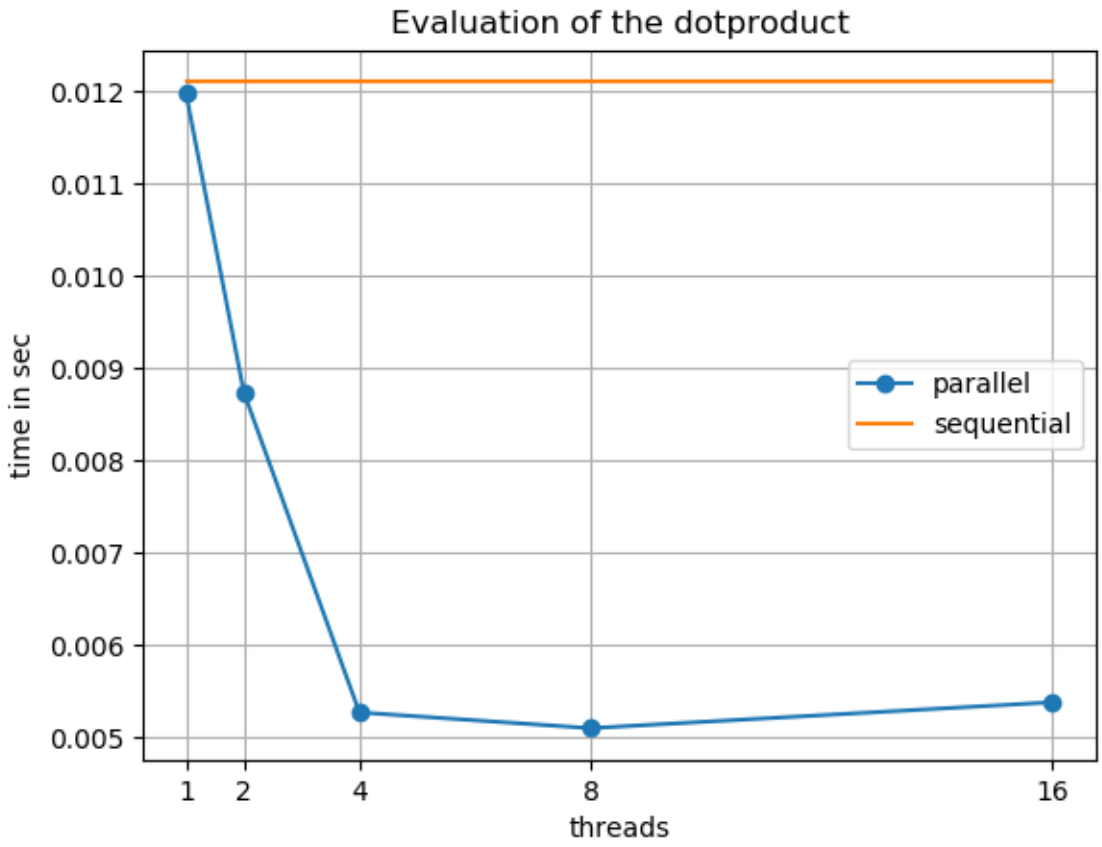
→ Task 2:

a) private or shared in omp parallel:
   1. i : private
   2. j : shared
   3. g1 : private
   4. g2 : shared

b) private or shared in foo:
   1. p : private
   2. g1 : shared
   3. g2 : shared

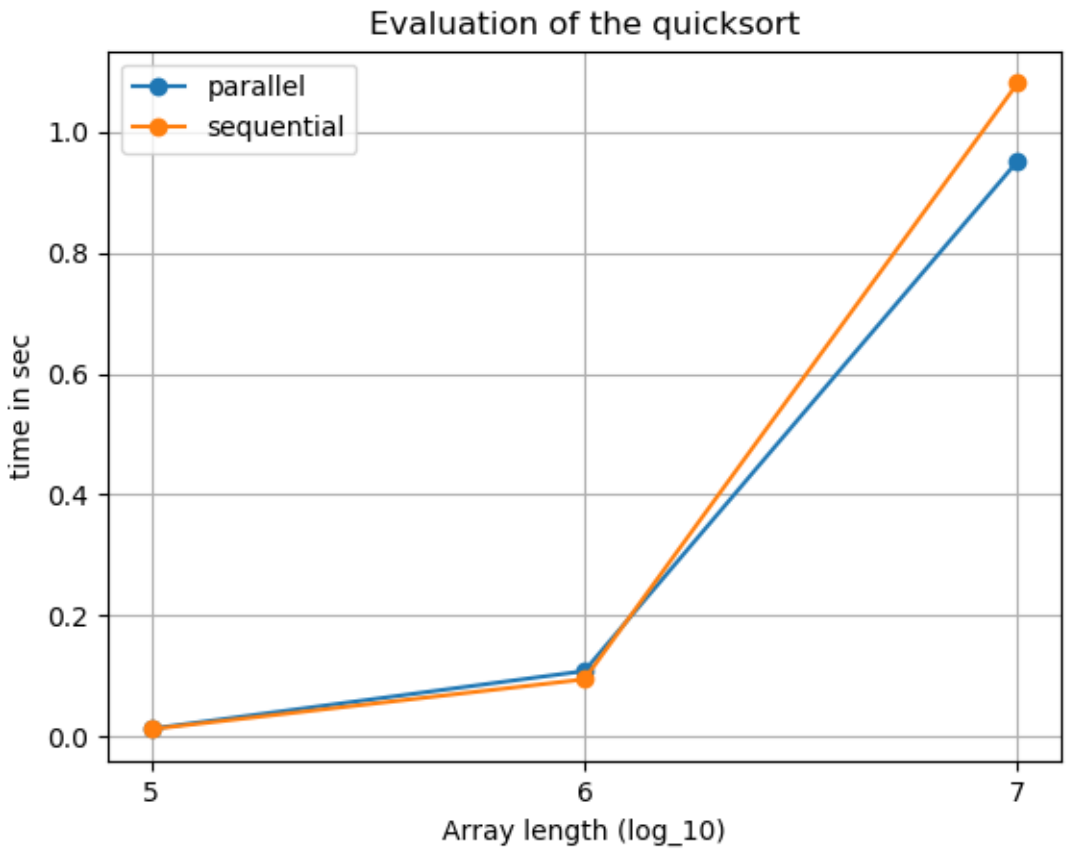
→ Task 3: Dotproduct

c) Laufzeitmessungen:



→ **Task 4:** Quicksort

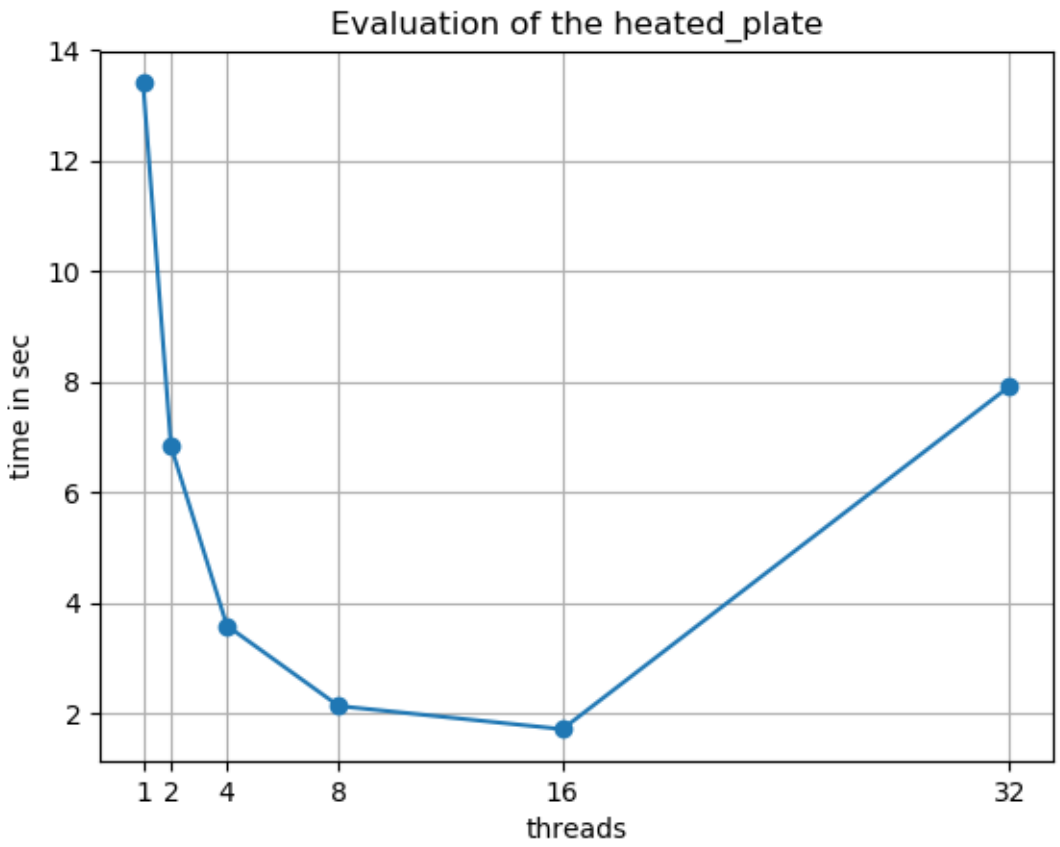
d) Laufzeitmessungen:



e) Die Funktion müsste parallel aufrufbar sein. Da sie ja ausgehend vom selben seed immer die gleiche Kette von Pseudo-zufälligen Nummern liefert, würde sonst jeder Thread dieselben „Zufallszahlen“ generieren. Daher wäre es essentiell, dass rand() auch parallel aufgerufen werden kann, ohne dadurch verfälschte/identische Rückgaben zu liefern.

→ **Task 5:** Heated-plate:

c) Laufzeitmessungen bei parallerer Ausführung mit unterschiedlicher Threadnummer



→ **Task 6:**

Der Aufruf „private“ erzeugt für jeden Thread eine eigene private variable a die jedoch **nicht initialisiert** ist („firstprivate“ übernimmt zuvor beinhaltete Werte mit in den parallelen Aufruf). Dies führt somit zu einem Fehler beim inkrementierten.

Allerdings wurde auf manchen unserer Geräte a automatisch mit 0 initialisiert, wodurch jeder Thread 1 ausgab. Dies geschah in Anlehnung an das oben genannte jedoch immer, unabhängig davon, wie a vor dem parallelen Aufruf initialisiert wurde. Ob ein Fehler ausgegeben wurde oder a automatisch mit 0 initialisiert hängt vermutlich an unterschiedlichen Versionen zusammen.