Course 02614: Assignment 2 — Evaluation January 18, 2023

Group #3: Gustav Gamst Larsen s180820
Lukas Leindals s183920
Simon Majgaard s184012

Comments:

Sequential implementations:

- both iterator schemes implemented correctly
- your scaling of the Frobenius norm with $1/N^3$ leads to the effect you observe in Fig .4 - did you ask for help on that?
- what do you observe for iter/sec for the two methods? You show something in the lower part of the same figure, but you don't discuss the difference (if any) between the two methods!

Parallel implementation:
Jacobi:

- your parallel Jacobi implementation is correct, but why did you choose the schedule(static,1)? Does this make sense? If not, why is this not good?
- I don't understand your Fig. 5: The upper part, static schedule, runs slower and has more barriers than the lower part, dynamic schedule. Two questions: did you maybe mix up the two cases? And which chunk sizes did you use in those cases?
- Why is the default static schedule (with a large chunk size, N/P, P the number of threads) supposed to be best here?
- thread placement and binding, in connection with first touch, is important for this problem - correct!
- Your Table 2: Which compiler did you use here? The mentioned compiler in the beginning of the report does not know "numa_domains", so this must have been a newer compiler!
- how many threads were used to get the timings in this table? To see the effect of those settings, you would need to plot speed-up for different settings, to see if there is any improvement.
- you get reasonable good speed-up for Jacobi (Fig. 6), for the sizes and compiler flags used! As it breaks down at 12 cores, i.e. 1 CPU socket, seems to indicate that your placement/binding settings didn't work! Maybe system sizes were too small, too, as N=256 can easily fit into the (two) L3 caches of the machine!
- I don't think your code is "wrong" - you were just looking at the right combinations and sizes
- no compiler optimization: "slow code scales better!" - but why?

Gauss-Seidel:

- your GS code looks correct
- GS scales better on a single CPU socket (up to 12 cores, here) - if the threads are placed "correctly", i.e. "close". Then the implementation benefits from sharing the L3 cache - which breaks down when going to the second socket.

Poor	Not quite adequate	Adequate	Good	Very good