Wednesday 01/11/12:

Ch. 11 Performance: Reference HPCF-2010-2

first day of class: Pages 1–4 ⇒

summary tables (pp. 3–4) for 1, 2, 4, ..., 64 nodes with 1, 2, 4, 6, 8 proc. per node

We start this way, because the gomb submission script needs the information in this way.

today: Pages 5–14 of this report ⇒ Introduce speedup and efficiency and their graphical representation, using trap. rule HW as example.

Also: Show some "best practices" for how to perform studies and how to collect their results ⇒ use this for future HW

Concretely: Ignore actual example in tech. rep., but view it as outline for approaching any example problem ⇒ Ignore Sec. 2 and from Sec. 3 only take away high-level ideas:

How is Sec. 3 structured?

Defines example precisely, as run in computer with all parameters, etc.

Report on small test case to confirm correctness of implementation first often in graphical form as solution plot, then in quantitative form by tabulating or plotting of errors ⟶ Table 3.1 ($\|u - u_h\|$, iter).

Then, here also included in the same table, report serial results to set the stage for parallel computing and to test feasibility of any study to be performed.

For example, we may want to run with progressively larger n to find one where the serial run is both feasible (less than 24 hours, e.g.) and non-trivial (more than 1 second, e.g.)

Also, ideally we want some problem that uses significant memory, so that parallelizing can show its power of solving a larger problem.

in Table 3.1, consider $n = 1024, \ldots, 16384$

Sec. 4 Performance Study:

Speedup $S_p = \frac{T_1}{T_p}$ where T_p is time on p procs

is the classical measure of efficiency.

If $S_p \cong p$ then the method/code/computer scale well.

Ideal case is that it takes $\frac{1}{p}$ time on p proc.

$\Rightarrow T_p = \frac{1}{p} T_1 \Rightarrow S_p \cong \frac{T_1}{T_p} = p$.

Why is this the best possible case, or is it?

Fundamentally, the advantage of splitting up work onto p procs is that each proc has less work to do and is faster

But as we involve more processes, there is more communication overhead in the program that the serial program does not have.
=> eventually, this overhead will negate any benefit of splitting the work.

▽ How might one get better than optimal speedup? This can happen if serial code is slower than it should be.
When you see better than optimal speedup in many cases of problem size n, then check serial code and its algorithm again.
→ power method example: order of i and j loops when accessing A_{ij} ▽
The code with the wrong loop order will be much slower than the code with the faster one, comparing serial to serial code.
The effect will be better speedup for the one and worse for the other.
△

Upshot: Realize that speedup is a relative measure ("how much faster"), but in the end, the absolute run time is really what matters!

Back to HPCF-2010-2, page 10:

We see speedup plot that indicates that only N=1024 does not scale well.

This corresponds to it taking only 29 sec. to start with.

We see here that there is more than one way to get, for instance, 16 processes in the MPI job, namely 2 nodes with 8 ppn each or 4 nodes with 4 ppn each, etc.

So, the report shows several tables with different possibilities.

Upshot: Today, it is no disadvantage to put as many processes as possible on each node!

Efficiency $E_p = \frac{S_p}{p}$ optimal value for $S_p = p$

1.0 = 100%. Plotting this quantity often brings out issues for small numbers of processes.

What is actually T_p?

To be comparable to T_1, we must solve a problem of same size as T_1 does.

(↳ does not work for all methods!)

(⇒ That is, must have same number of arithmetic operations.

Also, T_p must be wall clock time, not CPU time or similar, since only wall time includes communications and having those is integral to timing parallel code.

What is T_1 actually? Usually it is parallel code using 1 process, but in most code this is equivalent to serial code.