Parallel Programming

Timings

Prof. Paolo Bientinesi

HPAC, RWTH Aachen
pauldj@aices.rwth-aachen.de

WS16/17

Time

- **Wall time** or "wall-clock time" :
  real time between the beginning and the end of a computation

- $T_p(n)$:= Wall time to solve a problem of size n using p processes

  $T_p(n) = t_1 - t_0$
  t_0 : time when the first process starts its execution,
  t_1 : time when the last process completes its execution

- **CPU-time** or "core time" :
  cumulative time spent by all processes in a computation

Code – 1

- `time0.c`
  Cholesky factorization.
  No timings. Only correctness.

- `time1.c`
  Timings through `clock()`.
  Multithreading (via LAPACK/BLAS). CPU-time.

- `time2.c`
  Cycle accurate timer.
  Cycles, frequency. Wall time vs. CPU-time.

  Ver.2: Performance (# ops/sec), efficiency.

Code – 2

- `time3.c`
  GEMM as a triple loop.
  Terrible performance and efficiency. No parallelism.

- `time4.c`
  Timings breakdown: Malloc, init, compute, test.
  Scalability. Serial vs. parallel code. Ahmdal's law.

Performance

- **Performance**: Number of floating point operations per second performed while solving a given problem
- **Theoretical Peak Performance** (TPP): In ideal conditions, the highest number of floating point operations that a processor can perform in one second
- **Peak Performance** “Practical peak performance”: The performance attained by highly tuned matrix-matrix multiplication kernels (DGEMM). For instance, MKL and OpenBLAS.
- **Efficiency**: The ratio between the performance attained while solving a given problem and the TPP

Scalability

- **Speedup**: $S_p(n) := \dfrac{T_1(n)}{T_p(n)}$ Typically: $0 \leq S_p(n) \leq p$

  $T_1(n)$: best sequential code – possibly different algorithm

  If $S_p(n) > p$: "superlinear speedup" $\leftarrow$ rare

- **Parallel efficiency**: $E_p(n) := \dfrac{S_p(n)}{p}$ $0 \leq E_p(n) \leq 1$

- **Strong Scalability**: Behaviour of $T_k(n)$, as k increases.
  Fixed problem size, increasing number of processes.

- **Weak Scalability**: Behaviour of $T_k(m)$, as m and k increase so that the load per process stays constant. *Fixed load per process, increasing problem size and number of processes.*

Amdahl's law

Maximum possible speedup when only a portion of the code scales.

- T_{seq}: strictly sequential portion of the algorithm (in secs)
- T_{par}: parallel portion of the algorithm (in secs)
- $T_1(n) == T_{\text{seq}} + T_{\text{par}}$
- β: fraction of the algorithm that is strictly sequential
- $\beta == \dfrac{T_{\text{seq}}}{T_{\text{seq}} + T_{\text{par}}}$
- $T_p(n) == \beta T_1(n) + (1 - \beta)\, T_1(n)/p == T_1(n) \left(\beta + \frac{(1-\beta)}{p} \right)$
- $S_p(n) == \frac{T_1(n)}{T_1(n)\left(\beta + \frac{(1-\beta)}{p}\right)} == \frac{1}{\beta + (1-\beta)/p}$ $\qquad \lim_{p \to \infty} S_p(n) = 1/\beta$