Parallel Programming

Performance metrics

Prof. Paolo Bientinesi

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

WS18/19

High Performance and Automatic Computing

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

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 (or the PPP).

Speedup

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

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

- **What is $T_1(n)$?** Time of the best sequential code.
  NOT the time for the parallel code run with $p = 1$!!

  Note: The sequential code could possibly implement a different algorithm than the parallel one.

  Example: Eigenvalues of symmetric tridiagonal matrix
  - Alg.1: `dqds` cost: $O(n^2)$, BUT inherently sequential
  - Alg.2: `BX` cost: $O(n^3)$, BUT perfectly parallelizable

Speedup, Efficiency

- **What if p is large?** Then probably $T_1(n)$ is not obtainable, either because n is too large a problem to be solved sequentially, or because it would take too long to complete.

  In this case, $S_p(n) := \dfrac{T_{p_0}(n)}{T_p(n)}$, and $0 \leq S_p(n) \leq \dfrac{p}{p_0}$.

  $T_{p_0}(n)$ is used as a reference, possibly from a different code.

  Note: Excellent speedup (by itself) does not imply excellent algorithm.

- **Parallel efficiency**:

  $$E_p(n) := \frac{S_p(n)}{p} \qquad \left(\text{or } \; E_p(n) := \frac{S_p(n)}{p/p_0} \right) \qquad 0 \leq E_p(n) \leq 1$$

  Note: Excellent efficiency (by itself) does not imply excellent algorithm.

Scalability – pt.1

1) Scalability with respect to data (problem size). [no parallelism]

   Question: *How does the execution time increase as the problem size increases?*

2) Scalability with respect to resources (# of processors).

   Question: *How does the execution time increase as the amount of resources increases?*

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

  Example: $n = \bar{n}$; $p = 2^i$, with $i \in [10, \ldots, 14]$

  $\Rightarrow$ $T_{1024}(\bar{n})$, $T_{2048}(\bar{n})$, $T_{4096}(\bar{n})$, $T_{8192}(\bar{n})$, $T_{16384}(\bar{n})$

Code

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

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

- `time2a.c`
  Cycle accurate timer.
  Cycles, frequency. Wall time vs. CPU-time.

- `time2b.c`
  Performance (# ops/sec), efficiency.

- `time3.c`
  GEMM as a triple loop.
  Terrible performance and efficiency.

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)
- $T_1(n) == T_{\text{seq}} + T_{\text{par}}$
- $T_p(n) == T_{\text{seq}} + T_{\text{par}}/p$ ideal parallelisation
- $\beta == \dfrac{T_{\text{seq}}}{T_{\text{seq}} + T_{\text{par}}}$: fraction of the algorithm that is strictly sequential
- Speedup: $S_p(n) == \dfrac{T_1(n)}{T_p(n)} == \dfrac{T_{\text{seq}} + T_{\text{par}}}{T_{\text{seq}} + T_{\text{par}}/p} == \dfrac{1}{\beta + (1-\beta)/p}$
- Ideal Speedup: $\lim_{p \to \infty} S_p(n) = \dfrac{1}{\beta}$

Code

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

Scalability – pt.2

- **Weak Scalability**: Behaviour of $T_k(n)$, as n and k increase to keep the memory usage per process constant.
  Fixed memory load per process, increasing problem size and number of processes.

  Example: Algorithm $\mathcal{A}$; input: $n \in \mathbf{N}$
  Time complexity: $O(n^3)$ Space complexity: $O(n^2)$
  Reference: $\bar{n} = 100$, $\bar{p} = 16$.
  $\Rightarrow$ $T_{16}(100)$, $T_{64}(200)$, $T_{256}(400)$, $T_{1024}(800)$

Weak Scalability – Example #1

- Algorithm $\mathcal{B}(n)$
  Time($\mathcal{B}(n)$) = $O(n^2)$, Space($\mathcal{B}(n)$) = $3n$, Reference: $T_{\bar{p}}(\bar{n}) = t_0$.
  - $T_{??}(2\bar{n})$ — from problem size to number of processors
  - Space($\bar{n}$) = $3\bar{n}$ $\Rightarrow$ Mem/proc: $3\bar{n}/\bar{p}$ == *const*
  - Space($2\bar{n}$) = $6\bar{n}$ $\Rightarrow$ $6\bar{n}/??$ == *const* $\Rightarrow$ $?? = 2\bar{p}$
  - $T_{2\bar{p}}(2\bar{n}) = t_1$ $t_0 > / = / < t_1$?
  - Assumed: perfect scalability wrt size, strong scalability
    $t_1 = T_{2\bar{p}}(2\bar{n}) \approx 4\,T_{2\bar{p}}(\bar{n}) \approx 4\,T_{\bar{p}}(\bar{n})/2 = 2t_0$
  - $T_{2\bar{p}}(??)$ — from number of processors to problem size
  - Space($\bar{n}$) = $3\bar{n}$ $\Rightarrow$ Mem/proc: $3\bar{n}/\bar{p}$ == *const*
  - Space($??$) = $3\,??$ $\Rightarrow$ Mem/proc: $3\,??/2\bar{p}$ == *const* $\Rightarrow$ $?? = 2\bar{n}$

Weak Scalability – Example #2

- Algorithm $\mathcal{B}(n)$
  Time($\mathcal{B}(n)$): $O(n^2)$, Space($\mathcal{B}(n)$): n^2, Reference: $T_{\bar{p}}(\bar{n}) = t_0$.
  - $T_{??}(2\bar{n})$
  - Space($\bar{n}$) = $\bar{n}^2$ $\Rightarrow$ Mem/proc: $\bar{n}^2/\bar{p}$ == *const*
  - Space($2\bar{n}$) = $4\bar{n}^2$ $\Rightarrow$ $4\bar{n}^2/?? == const$ $\Rightarrow$ $?? = 4\bar{p}$
  - $T_{4\bar{p}}(2\bar{n}) = t_1$ $t_0 > / = / < t_1$?
  - Assumed: perfect scalability
    $t_1 = T_{4\bar{p}}(2\bar{n}) \approx 4\,T_{4\bar{p}}(\bar{n}) \approx 4\,T_{\bar{p}}(\bar{n})/4 = t_0$
  - $T_{2\bar{p}}(??)$
  - Space($\bar{n}$) = $\bar{n}^2$ $\Rightarrow$ Mem/proc: $\bar{n}^2/\bar{p}$ == *const*
  - Space($??$) = $??^2$ $\Rightarrow$ Mem/proc: $??^2/2\bar{p} == const$ $\Rightarrow$ $??^2 == 2\bar{p}\bar{n}^2/\bar{p}$
    $\Rightarrow$ $?? == \bar{n}\sqrt{2}$