Parallel and High Performance Computing

Oral presentation

Alexandre Youssef

January 13, 2023

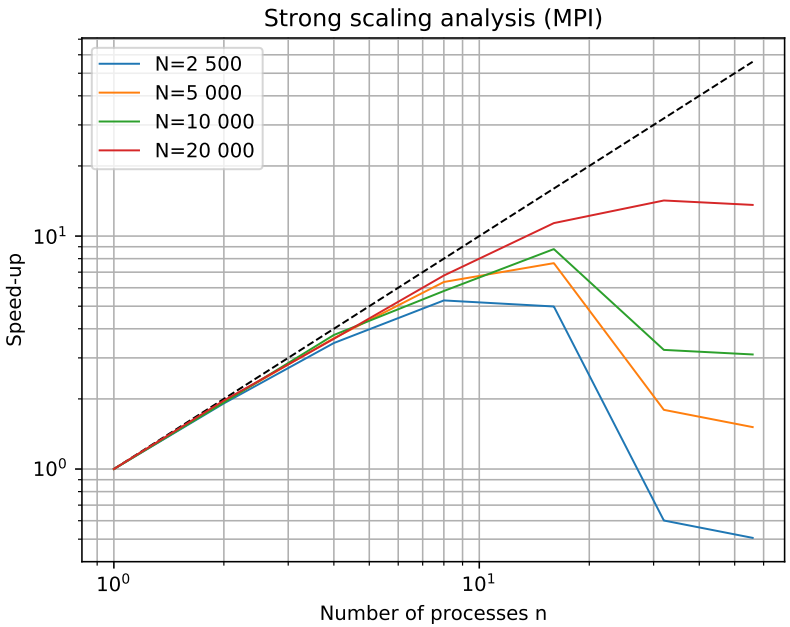
EPFL

Table of contents

EPFL

Strong scaling with Amdhal's prediction (MPI)

Amdahl's law : $S(n) = \frac{1}{(1-p)+\frac{p}{n}}$ with n processes and a fraction p of the code that is (perfectly) parallel. Choice : parallelize the entire CG algorithm. → sequential part $f = 1 - p =$ reading the matrix



(a) Theoretically, matrix reading is in $\mathcal{O}(nz)$. $p = 1 - \frac{nz}{2nz+3N} \approx 1$ for a sparse matrix.

(b) Compute $\frac{t_{\text{read}}}{t_{\text{tot}}}$ for large nz.
$p = 1 - \frac{0.035}{22.081} = 100\% - 0.15\% \approx 100\%$.

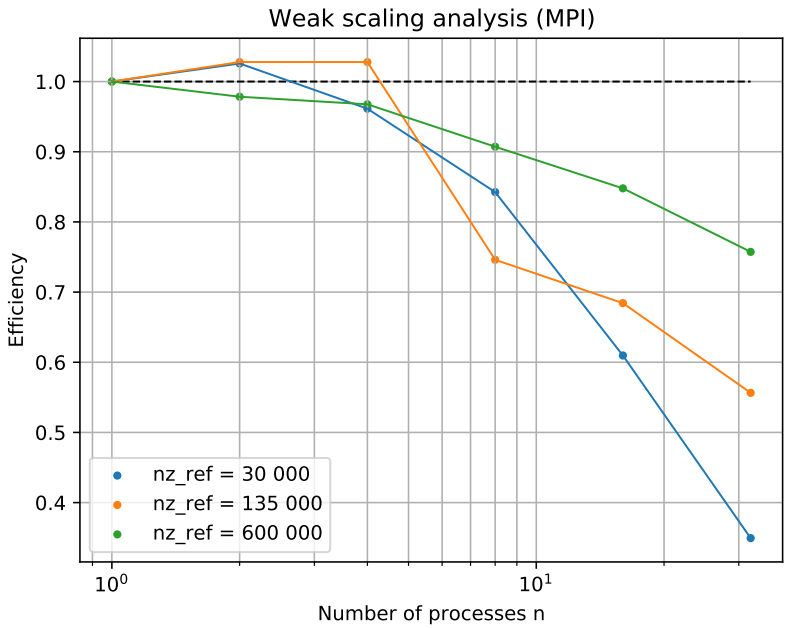
Amdahl's law predicts $S(n) = n$ (perfect scaling). Not the case here, why ?

From the graph, two main results ($N \propto nz$) :

1. When $N \nearrow$, better scaling. Amount of computation of the single processes is big enough compared to the communication.
2. When using 2 nodes ($p > 28$), drop of the speed-up due to latency.

Weak scaling with Gustafson's prediction (MPI)

Gustafson's law : $S(n) = n + f(1-n)$. Theoretically, efficiency $\frac{S(n)}{n}$ should be close to 100%. We now consider a fixed workload per processor and check for the result.



Operations : reading + 1 matrix vector product + 3 vector additions : $\mathcal{O}(2nz + 3N)$.

Assuming $N \propto nz$, linear variation of the total amount of work with N.

On the graph : three different initial matrix sizes, then truncated as $p \searrow$.

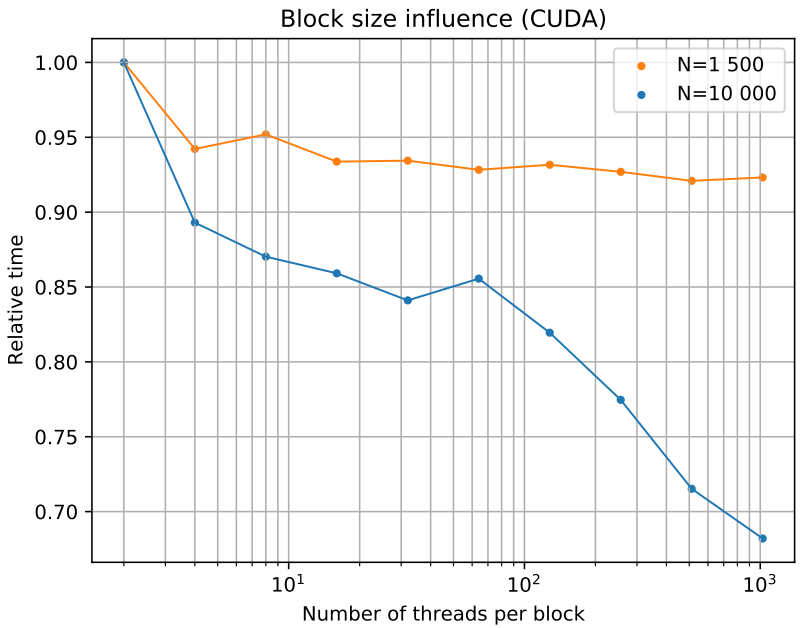
⚠Check that the total number of operations remains the same when truncating

From the graph, two main results :

1. Drop of the efficiency due to communication overheads.
   Indeed, 1 MPI_Allgatherv() and 2 MPI_Allreduce() at each iteration.
2. Better efficiency as $nz \searrow$. The computational cost is larger than the communication cost as nz increases.

Grid and Block sizes influence (CUDA)

Single parameter to vary : the block size. The grid size is fixed, depending on a matrix-vector multiplication (nz) or a vector-vector addition (N). In the matrix-vector product, use of atomicAdd() : avoid conflicts but time consuming !



Test with $N = 1500$ (less conflicts) and $N = 10000$ (more conflicts).

For more than 32 threads per blocks, warps are filled + enough blocks to saturate the GPU → convergence for $N = 1500$.
If there is more conflicts ($N = 10000$), the graph will keep decreasing because the amount of conflicts still decreases.

Two additional comments :

1. For the Laplacian matrix example, the sequential code is faster : the thread conflicts imply a longer execution time for the CUDA version.
2. However, this is not the case anymore as $nz \nearrow$.
   Example : for $nz = 10^6$, 208s vs 2.71s with 1024 threads per block !