Theoretical Computational Speed-Up in SPARK

by Dimitri Curtil (SRG/LBNL)

Typical measures of computational efficiency are:

1. the number of function evaluations,
2. the number of iterations required to solve the nonlinear system of equations, and
3. the computational cost per iteration.

It is important to keep in mind that the overall cost of solving the system of equations depends on the number of iterations required to achieve some prescribed precision in all the unknown variables. Since the solution sequence derived by SPARK in a strongly-connected component is different than the one used by conventional solvers, the number of iterations until convergence might differ and therefore impact the overall computational cost. However, this discrepancy cannot be quantified theoretically as it depends inherently on the equation set being solved. Therefore, we cannot use the number of iterations as a fair comparative measure.

To assess the theoretical computational speed-up obtained with the SPARK graph-theoretical analysis we will derive the metrics 1 and 3 for a system of n equations and compare them with the same metrics typical of conventional solvers for a system of the same size.

Solving a System of Nonlinear Equations

Solving a system of nonlinear equations with n equations and n unknowns consists in finding the root x_* assuming that the function F with all the equations is continuously differentiable:

$$\begin{cases} Given\ F : \mathbb{R}^n \to \mathbb{R}^n \\ Find\ x_* \in \mathbb{R}^n\ such\ that\ F(x_*) = 0 \end{cases} \tag{1.1}$$

This problem is typically solved using the Newton method. At each iteration k, the Newton method consists in solving the following linear system:

$$\begin{cases} \dfrac{\partial F(x_k)}{\partial x_k} \cdot \Delta x_k = -F(x_k) \\ x_{k+1} = x_k + \Delta x_k \end{cases} \tag{1.2}$$

Structure of a Strongly-connected Component in SPARK

A strongly-connected component in SPARK is a subsystem of n nonlinear equations and n unknowns. Following the cut set reduction algorithm the n unknowns are split between n_u non-break unknowns u and n_v break unknowns v, whereby $n = n_u + n_v$.

$$\begin{cases} \mathbb{R}^{(n_u+n_v)} \to \mathbb{R}^{(n_u+n_v)} \\ (u,v) \mapsto F(u,v) = \begin{bmatrix} G(u,v) \\ H(u,v) \end{bmatrix} \end{cases} \tag{1.3}$$

The subsystem of equations $G(u,v) = u - g(u,v)$ is ordered in such a way that it can be solved using forward substitution when the break variables v are known. Here, the functions $g(u,v)$ represent the actual inverses matched with the non-break variables u.

$$\frac{\partial g(u,v)}{\partial u} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ g_{2,1} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ g_{n_u,1} & \cdots & g_{n_u,n_u-1} & 0 \end{bmatrix} \tag{1.4}$$

Because of graph-theoretical derivation of the small cut set, only the n_v equations forming the vector $H(u,v)$ need to be solved simultaneously for the n_v break variables using an iterative solution method, e.g., the Newton method.

At each iteration k, the Newton method consists in solving the following linear system for the reduced system of equations H:

$$\begin{cases} \dfrac{\partial H(u_k,v_k)}{\partial v_k} \cdot \Delta v_k = -H(u_k,v_k) \\ v_{k+1} = v_k + \Delta v_k \end{cases} \tag{1.5}$$

Computational Cost per Iteration

The computational cost of a Newton iteration, excluding function and derivative evaluation, is invariably determined by the linear algebra, [1]. The arithmetic cost of the factorization of a dense matrix of size $n \times n$ is typically a small multiple of n^3.[1]

Thus, the computational cost of a Newton iteration in a strongly-connected component is proportional to n_v^3.

[1] The arithmetic cost of the matrix factorization is proportional to n^2 for secant methods.

Number of Function Evaluations

The number of function evaluations needed to compute the residual function used in the Newton iterations (1.2) and (1.5) is identical for the conventional solver and the SPARK solver. At each iteration it is: $n = n_u + n_v$

The number of function evaluations needed to compute the finite differences to form the Jacobian matrix is $n \cdot n = n^2$ for the function F in (1.2) whereas it is $n_v \cdot n$ for the function H in (1.5). Depending on the strategy implemented in the actual nonlinear solver the Jacobian matrix might not be re-computed at each iteration.

Summary

Speed-up in each strongly-connected component

To estimate the computational speed-up achieved with the SPARK approach, we need to introduce the notion of the reduction ratio as the ratio of the number of break variables over the number of unknown variables:

$$r = \frac{n_v}{n} \tag{1.6}$$

The reduction ratio is smaller or equal to 1. It is equal to 1 when the identified cut set is exactly the set of all unknowns.

The theoretical speed-up is estimated as the ratio of the SPARK metric over the corresponding metric for a conventional solver.

	Conventional solver	***SPARK solver***	***Speed-up***
Computational cost per iteration	$\mathrm{O}\left(n^3\right)$	$\mathrm{O}\left(n_v{}^3\right)$	$\mathrm{O}\left(r^3\right)$
Number of function evaluations to compute residuals	n	n	1
Number of function evaluations to compute finite-differences	n^2	$n_v \cdot n$	r

At each iteration, the theoretical speed-up achieved by the SPARK solution technique is as high as $\mathrm{O}\left(r^3\right)$ in each strongly-connected component. This comparison is only true when using linear solution methods for dense matrices.

Impact of using sparse solution techniques

When using sparse solution techniques, the speed-up factor will be less drastic as the computational cost per iteration becomes proportional to the number of non-zero entries raised to some exponent usually smaller than 3, typically $\mathrm{O}\left(n^2\right)$.

To be fair, it is not straight-forward to derive a theoretical speed-up factor since the Jacobian matrix for the function H might intrinsically be less sparse than the one for the original function F. However, when solving "large"[2] strongly-connected components, experience has shown that SPARK benefits drastically from using a sparse solution technique.[3] The more equations there are in the component the less likely they are to all depend on all the unknown variables, therefore still producing a very sparse Jacobian matrix.[4] Although the sparsity pattern in the Jacobian matrix for the SPARK approach is likely to be different than the one observed in the Jacobian matrix for the conventional solver, it is still significant enough when n_v is big enough.

Impact of the strong component decomposition

The graph-theoretic analysis in SPARK also performs a strong component decomposition, whereby the original problem is further decomposed into smaller sub-problems that are solved in sequential order. This can represent drastic further computational speed-up over the conventional approach that solves the entire set of equations at once.

If SPARK identifies N strongly-connected components and M "explicit" components,[5] the speed-up K_{ccpi} in the computational cost per iteration to solve the linear system is:

$$K_{ccpi} = \frac{\sum_{i=1}^{N} \mathrm{O}\left(c_i^{\,3}\right)}{\mathrm{O}\left(n^3\right)} \tag{1.7}$$

where

c_i size of cut set (break variables) in strongly connected component i

n total number of unknown variables in problem ($\sum_{i=1}^{N} c_i \le n$)

This speed-up is only fully achieved if we further assume that the same number of iterations is required to solve each nonlinear problem in the strongly-connected components as it is in the full problem.

A loose estimate $\bar{K}_{ccpi}$ of the speed-up in Equation (1.7) can be derived using the size of the largest cut set $c_{\max}$:

$$\bar{K}_{ccpi} \le \frac{N \cdot \mathrm{O}\left(c_{\max}^{\,3}\right)}{\mathrm{O}\left(n^3\right)} \tag{1.8}$$

[2] With at least 100 break variables.

[3] The sparse solution technique implemented in the SPARK linear solver is based on the UMFPACK library (http://www.cise.ufl.edu/research/sparse/umfpack/).

[4] The Jacobian matrix for a function H with $n_v > 100$ is typically over 90% sparse!

[5] An explicit component does not contain any break variables, therefore it can be solved by forward substitution. This means that the computational cost is exactly n_u function evaluations.

Finally, the SPARK graph-theoretic approach also delivers a decrease in the number of function evaluations required to compute the finite-differences. This speed-up K_{fefd}, although of lower order than the one achieved in the computational cost per iteration, can also be significant if the functions comprising the nonlinear problem are expensive to evaluate.

$$K_{fefd} = \frac{\sum_{i=1}^{N} r_i \cdot n_i^2}{n^2} \tag{1.9}$$

where

r_i cut set reduction ratio achieved in the strongly connected component i

n_i number of unknown variables in the strongly connected component i

n total number of unknown variables in problem ($\sum_{i=1}^{N} n_i \le n$)

References

[1] Dennis J. E. and Schnabel R. B. (1996), "Numerical Methods for Unconstrained optimization and Nonlinear Equations", SIAM Classics in Applied Mathematics 16, Englewood Cliffs, N.J.