Pipelined iterative methods

Maya Neytcheva

NLA, Feb-March, 2021

Pipelined versions of Krylov subspace iteration methods

Main idea: collect the scalar products in one place, to reduce communication in parallel computations.

Standard CG

Given A, $\mathbf{b}$ and an initial guess $\mathbf{x}^{(0)}$.

```
     $\mathbf{r}^{(0)} = A\mathbf{x}^{(0)} - \mathbf{b}, \quad \mathbf{g}^{(0)} = \mathbf{r}^{(0)}$
1    for $k = 0, 1, \cdots$ until convergence
2        $\tau_k = \frac{(\mathbf{r}^{(k)}, \mathbf{r}^{(k)})}{(A\mathbf{g}^{(k)}, \mathbf{g}^{(k)})}$
3        $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \tau_k \mathbf{g}^{(k)}$
4        $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} + \tau_k A\mathbf{g}^{(k)}$
5        $\beta_k = \frac{(\mathbf{r}^{(k+1)}, \mathbf{r}^{(k+1)})}{(\mathbf{r}^{(k)}, \mathbf{r}^{(k)})}$
6        $\mathbf{g}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta_k \mathbf{g}^{(k)}$
7    endfor
```

$\mathbf{r}^{(k)}$ – iteratively computed residuals
$\mathbf{g}^{(k)}$ – search directions

Pipelined unpreconditioned CG

Given A, $\mathbf{b}$ and an initial guess $\mathbf{x}^{(0)}$.

```
      $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}, \mathbf{w}^{(0)} = A\mathbf{x}^{(0)}$
1     for $k = 0, 1, \cdots$ until convergence
2         $\gamma_k = (\mathbf{r}^{(k)}, \mathbf{r}^{(k)})$
3         $\delta = (\mathbf{w}^{(k)}, \mathbf{r}^{(k)})$
4         $\mathbf{q} = A\mathbf{w}^{(k)}$
5     if $k > 0$,
6             $\beta_k = \gamma_k/\gamma_{k-1}; \alpha_k = \gamma_k/(\delta - \beta_k\gamma_k/\alpha_{k-1})$
7         else
8             $\beta_k = 0; \alpha_k = \gamma_k/\delta$ :
9         endif
10        $\mathbf{z}^{(k)} = \mathbf{q} + \beta_k \mathbf{z}^{(k-1)}$
11        $\mathbf{s}^{(k)} = \mathbf{w}^{(k)} + \beta_k \mathbf{s}^{(k-1)}$
12        $\mathbf{p}^{(k)} = \mathbf{r}^{(k)} + \beta_k \mathbf{p}^{(k-1)}$
13        $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$
14        $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{s}^{(k)}$
15        $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \mathbf{z}^{(k)}$
16    endfor
```

Pipelined preconditioned CG

```
$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}, \mathbf{u}^{(0)} = M^{-1}\mathbf{r}^{(0)}, \mathbf{w}^{(0)} = A\mathbf{u}^{(0)}$
for $k = 0, 1, \cdots$ until convergence
    $\gamma_k = (\mathbf{r}^{(k)}, \mathbf{u}^{(k)})$
    $\delta = (\mathbf{w}^{(k)}, \mathbf{u}^{(k)})$
    $\mathbf{m} = M^{-1}\mathbf{w}^{(k)}$
    $\mathbf{n} = A\mathbf{m}$
    if $k > 0$,
        $\beta_k = \gamma_k/\gamma_{k-1}; \alpha_k = \gamma_k/(\delta - \beta_k\gamma_k/\alpha_{k-1})$
    else
        $\beta_k = 0; \alpha_k = \gamma_k/\delta$ :
    endif
    $\mathbf{z}^{(k)} = \mathbf{n} + \beta_k\mathbf{z}^{(k-1)}$
    $\mathbf{q}^{(k)} = \mathbf{m} + \beta_k\mathbf{q}^{(k-1)}$
    $\mathbf{s}^{(k)} = \mathbf{w}^{(k)} + \beta_k\mathbf{s}^{(k-1)}$
    $\mathbf{p}^{(k)} = \mathbf{u}^{(k)} + \beta_k\mathbf{p}^{(k-1)}$
    $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{p}^{(k)}$
    $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k\mathbf{s}^{(k)}$
    $\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} - \alpha_k\mathbf{q}^{(k)}$
    $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k\mathbf{z}^{(k)}$
endfor
```

Convergence rate, computational cost per iteration

s-step Krylov subspace iteration methods

Erin Carson
The adaptive *s*-step Conjugate Gradient Method
SIAM J. MATRIX ANAL. APPL., Vol. 39 2018, pp. 1318–1338
See the paper.

Erin C. Carson, Miroslav Rozloznik, Zdenek Strakos, Petr Tichy, Miroslav Tuma
The numerical stability analysis of pipelined conjugate gradient methods: historical context and methodology
SIAM J. Sci. Comput., 40(5), A3549–A3580, 2018
Conclusions: "... We give an expression for the residual gap which is applicable to any such variant, including the pipelined CG method of Ghysels and Vanroose [18]. We show that, similar to 3-term recurrence CG, any variant of CG which uses an auxiliary recurrence for Api in updating the residual vector can suffer from amplifications of local rounding errors due to large residual oscillations.
... modifications borne out of efforts to minimize time per iteration. While an important piece of the puzzle, one must be cautious about using time per iteration as a metric of efficiency. Alone, this fails to capture the whole picture of the method's effectiveness within the context of a scientific application. If the modifications made to the method cause a convergence delay with a greater effect than the per-iteration performance improvement, the modified method may actually be slower than the standard approach in a practical setting. If the modifications to the method cause a significant decrease in attainable accuracy, then the method may have no use in some scientific applications, making any gains in iteration speed for naught."

Bottom line: be precautious when trading numerical efficiency for parallel efficiency!
There should be a theoretical support for the algoritmic changes to ensure the quality of the numerically computed solutions.