Three Week Presentation

Matrix Free Pre-Conditioner for Exascale Computing

Bolliger Matteo

July 4, 2024

1. Problem
2. Current work status
3. Task
4. Next steps

Problem

$$Ax = b \tag{1}$$

Iterative solvers

- Conjugate Gradient

Preconditioners

$$M^{-1}Ax = M^{-1}b \tag{2}$$

- lower condition number of Matrix A
- Faster convergence of iterative method
- Matrix-free

Exascale Computing

- Ever-increasing depth of hierarchy
- Scalability
- Performance portability

Preconditioned Conjugate Gradient (PCG)

Algorithm 1 Preconditioned Conjugate Gradient

Provided Parameters: F_A , $F_{M^{-1}}$, $\mathbf{b}$, $\mathbf{x}$, ε $\triangleright$ $\mathbf{x}$ is an initial guess, ε is the relative tolerance

$i \leftarrow 0$

$\mathbf{r} \leftarrow \mathbf{b} - F_A(\mathbf{x})$

$\mathbf{d} \leftarrow F_{M^{-1}}(\mathbf{r})$

$\delta_{\text{new}} \leftarrow \mathbf{r}^T\mathbf{d}$

$\delta_0 \leftarrow \delta_{\text{new}}$

while $i < i_{\max}$ and $\delta_{\text{new}} > \epsilon^2\delta_0$ **do**

- $\mathbf{q} \leftarrow F_A(\mathbf{d})$
- $\alpha \leftarrow \frac{\delta_{new}}{d^T q}$
- $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$
- $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{q}$
- $\mathbf{s} \leftarrow F_{M^{-1}}(\mathbf{r})$
- $\delta_{old} \leftarrow \delta_{new}$
- $\delta_{new} \leftarrow \mathbf{r}^T\mathbf{s}$
- $\beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$
- $\mathbf{d} \leftarrow \mathbf{s} + \beta\mathbf{d}$
- $i \leftarrow i + 1$

end while

return x

Gauss-Seidel

- Problem

$$Ax = b \qquad A = L + D + U \tag{3}$$

$$(L + D)x = b - Ux \tag{4}$$

- Iteration (compact form)

$$x^{(k+1)} = (L + D)^{-1}(b - Ux^{(k)}) \tag{5}$$

2-step Gauss-Seidel

- Outer Iteration (compact form)

$$x^{(k+1)} = (L+D)^{-1}(b - Ux^{(k)}) \tag{6}$$

- Inner Iteration (forward sweep)

$$x^{(k)} = g \qquad r = b - Ux^{(k)} \tag{7}$$

$$(L+D)g = r \tag{8}$$

$$Dg = r - Lg \tag{9}$$

$$g^{(j+1)} = D^{-1}(r - Lg^{(j)}) \tag{10}$$

PAUL SCHERRER INSTITUT

IPPL Implementation: 2-step Gauss-Seidel

Algorithm 4 2-step Symmetric Gauss-Seidel

Provided Parameters: $F_{D^{-1}}$, F_L , F_U , $\mathbf{b}$, k_{inner} , k_{outer}

$\mathbf{x} \leftarrow \mathbf{0}$

for $k \leftarrow 1, k_{\text{outer}}$ **do**

  $\mathbf{r} \leftarrow \mathbf{b} - F_U(\mathbf{x})$

  for $j \leftarrow 1, k_{\text{inner}}$ **do**

    $\mathbf{r}_{\text{inner}} \leftarrow \mathbf{r} - F_L(\mathbf{x})$

    $\mathbf{x} \leftarrow F_{D^{-1}}(\mathbf{r})$

  end for

  $\mathbf{r} \leftarrow \mathbf{b} - F_L(\mathbf{x})$

  for $j \leftarrow 1, k_{\text{inner}}$ **do**

    $\mathbf{r}_{\text{inner}} \leftarrow \mathbf{r} - F_U(\mathbf{x})$

    $\mathbf{x} \leftarrow F_{D^{-1}}(\mathbf{r})$

  end for

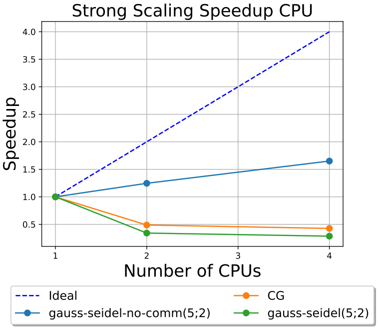
end for

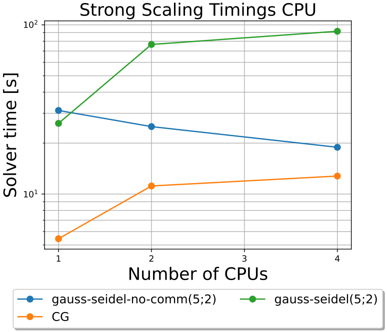
return $\mathbf{x}$

IPPL Implementation: Upper Laplace $F_U(x)$

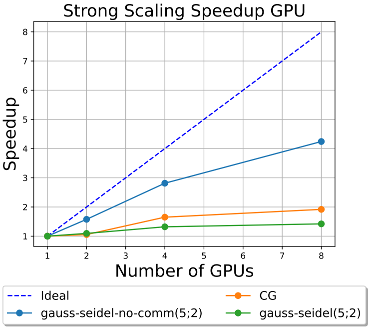
```
template <typename... Idx>
KOKKOS_INLINE_FUNCTION auto operator()(const Idx... args) const {
    using index_type = std::tuple_element_t<0, std::tuple<Idx...>>;
    using T          = typename E::Mesh_t::value_type;
    T res            = 0;

    for (unsigned d = 0; d < dim; d++) {
        index_type coords[dim]        = {args...};
        const int global_index        = coords[d] + ldom_m[d].first() - nghosts_m;
        const int size                = domain_m.length()[d];
        const bool left_boundary      = (global_index == 0);
        const bool not_right_boundary = (global_index != size - 1);

        coords[d] -= 1;
        auto&& left = apply(u_m, coords);

        coords[d] += 2;
        auto&& right = apply(u_m, coords);

        // left_boundary and not_right_boundary are boolean values
        // Because of periodic boundary conditions we need to add this boolean mask to
        // obtain the upper triangular part of the Laplace Operator
        res += hvector_m[d] * (left_boundary * left + not_right_boundary * right);
    }
    return res;
}
```

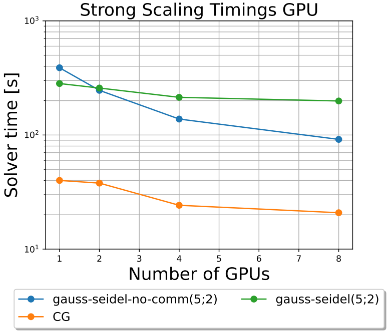
Current work status: CPU





Current work status: GPU





Task

- Identify limitations of existing implementation
- Improve scalability from found shortcomings
- Extend solver with symmetric successive over-relaxation (SSOR)

PAUL SCHERRER INSTITUT

SSOR

$$M_{SSOR} = \frac{1}{\omega(2-\omega)} \cdot (D+\omega L)D^{-1}(D+\omega U) \tag{11}$$

PAUL SCHERRER INSTITUT

Next steps

- Profiling of PCG
  - add timers to preconditioner
  - identify Kernels which may present a bottleneck
- Optimize Matrix free operators

References

[1] B. Schreiner, A Performance Portable and Matrix-Free Preconditioner for the Conjugate Gradient Solver (2024)

[2] Gen LI, Chun-an TANG, Lian-chong LI, High-effciency improved symmetric successive over-relaxation preconditioned conjugate gradient method for solving large-scale finite element linear equations (2013)