

Parallel Conjugate Gradient Method with Stencil-Based Matrix-Vector Multiplication

Algorithm The Conjugate Gradient (CG) Algorithm

Define A and b based on the input parameter n

Define maximum iterations $imax$ and tolerance error $rmax$

Initialize $u = 0$, $g = -b$, $d = b$

$q_0 = g^T g$

for $it = 1, 2, \dots$ **until** $it == imax$ or $|b - Au| < rmax$ **do**

$q = Ad$

$\tau = q_0 / (d^T q)$

$u = u + \tau d$

$g = g + \tau q$

$q_1 = g^T g$

$\beta = q_1 / q_0$

$d = -g + \beta d$

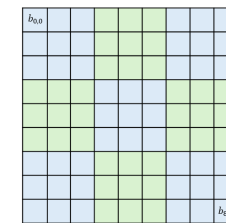
$q_0 = q_1$

end for

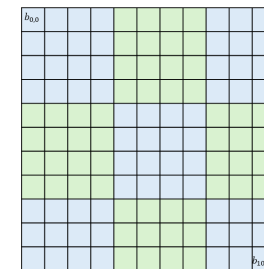
return u , the approximation solution

- ▷ Step length
- ▷ Approximate solution
- ▷ New residual
- ▷ Improvement
- ▷ New search direction

Initialize b



$n = 9$



$n = 11$ is not divisible by \sqrt{p}

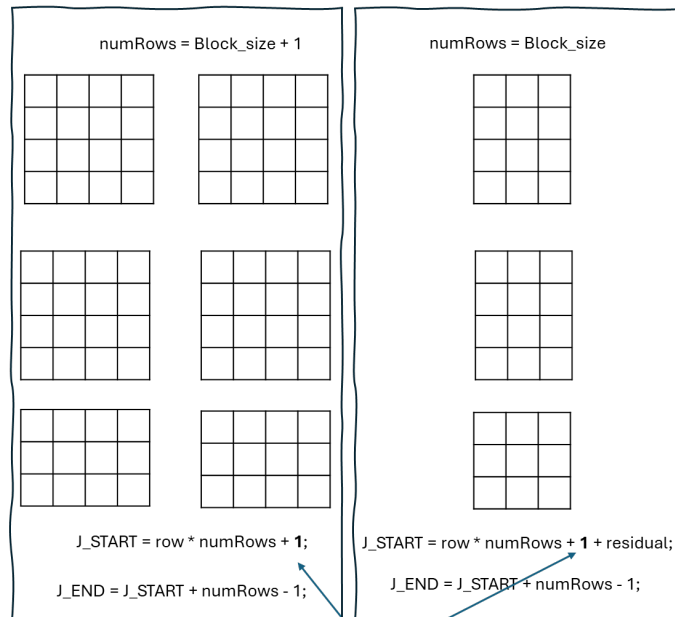


P0	P1	P2
P3	P4	P5
P6	P7	P8

Distribute mesh points among p processors

Initialization

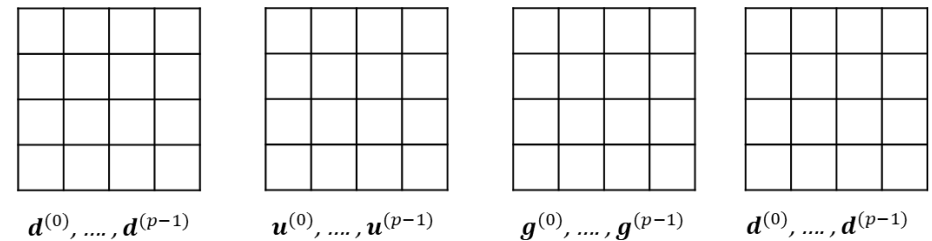
block_size = n / \sqrt{p}
 residual = n % \sqrt{p}
 If row < residual:
 get an extra row



The index i,j to calculate b_ij starts from 1

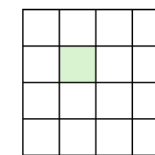
```
double x, y;
for (int i = 0; i < numRows; i++) {
    for (int j = 0; j < numCols; j++) {
        x = (J_START + i) * h;
        y = (J_START + j) * h;
        b[i][j] = 2 * h * h * (x * (1-x) + y * (1-y));
    }
}
```

Initialize $\mathbf{u} = \mathbf{0}$, $\mathbf{g} = -\mathbf{b}$, $\mathbf{d} = \mathbf{b}$, and $\mathbf{q}_0 = \mathbf{g}^T \mathbf{g}$



These vectors have the same shapes on each process, since all the basis operations are element-wise operations.

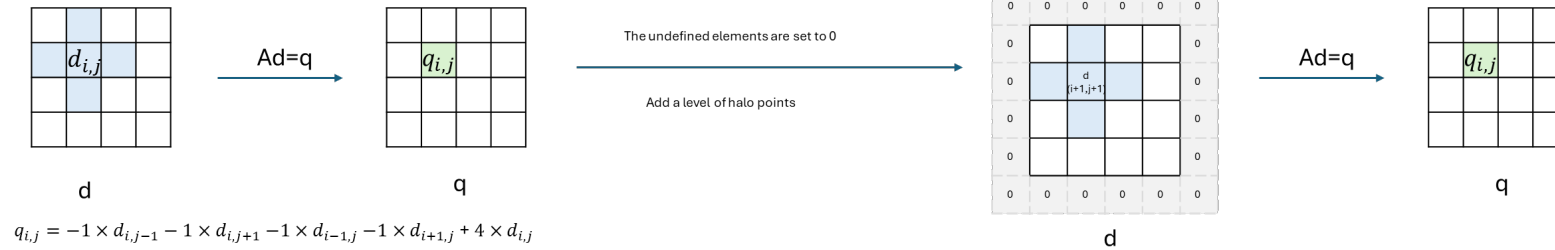
$$s^{(i)} = \sum_j q_{i,j}^{(i)} q_{i,j}^{(i)}$$



MPI_Allreduce()

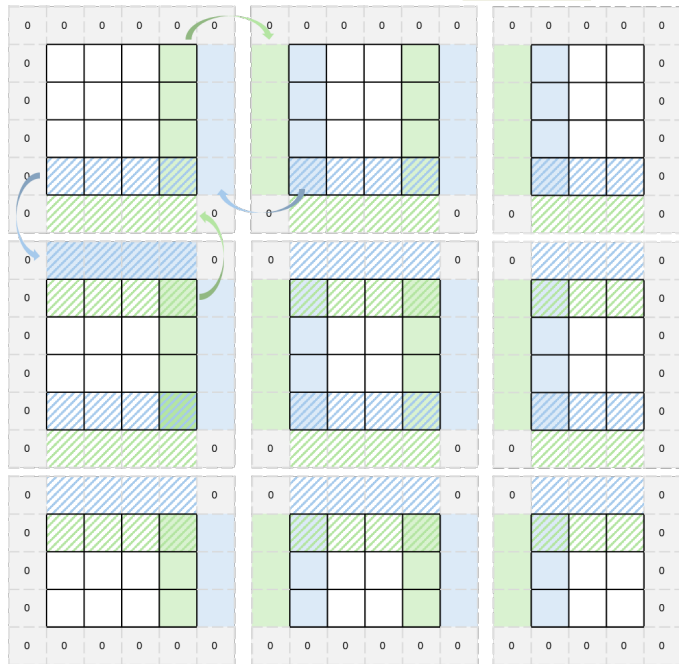
$$\text{Scalar product: } q_0 = \mathbf{q}^T \mathbf{q} = \sum_{i=0}^{p-1} s^{(i)}$$

Iteration Process

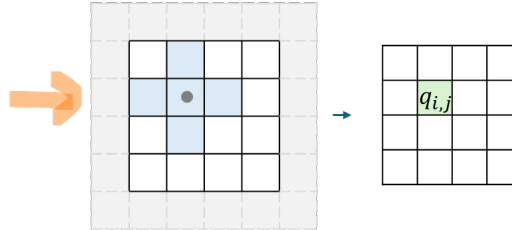


Step1. Exchange Boundary Data

Four directions: up, down, left, right, use `MPI_Sendrecv()`



index of elements of d shift by 1
 $q_{i,j} = -1 \times d_{i+1,j} - 1 \times d_{i+1,j+2} - 1 \times d_{i,j+1} - 1 \times d_{i+2,j+1} + 4 \times d_{i+1,j+1}$



Step2. $Ab^{(i)} = q^{(i)}, i = 0, 1, p-1$

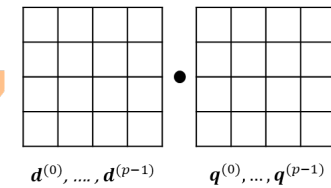
Repeat 200 times

Step4. Vector updates: $u = u + \tau d, g = g + \tau q$

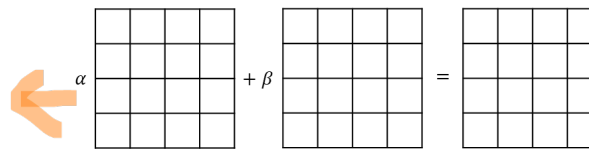
Step3. Scalar product: $s = d^T q = \sum_{i=0}^{p-1} s^{(i)}, \tau = q_0/s$

`MPI_Allreduce()`

$$s^{(i)} = \sum_i \sum_j d_{i,j}^{(i)} q_{i,j}^{(i)}$$

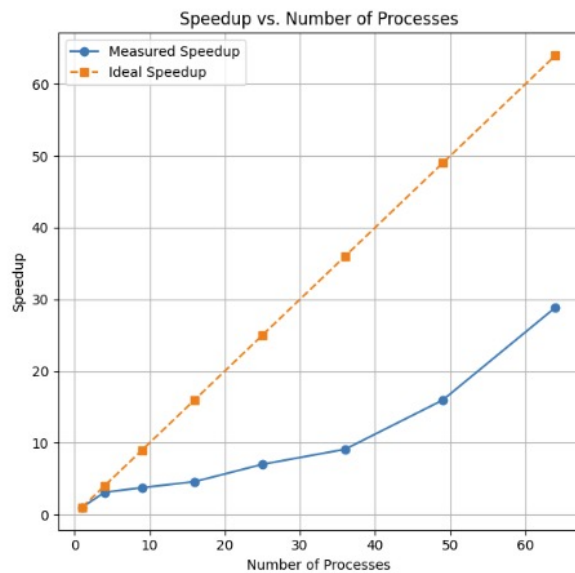


Step5. Scalar product: $q_1 = d^T q = \sum_{i=0}^{p-1} s^{(i)}, \beta = q_1/q_0$

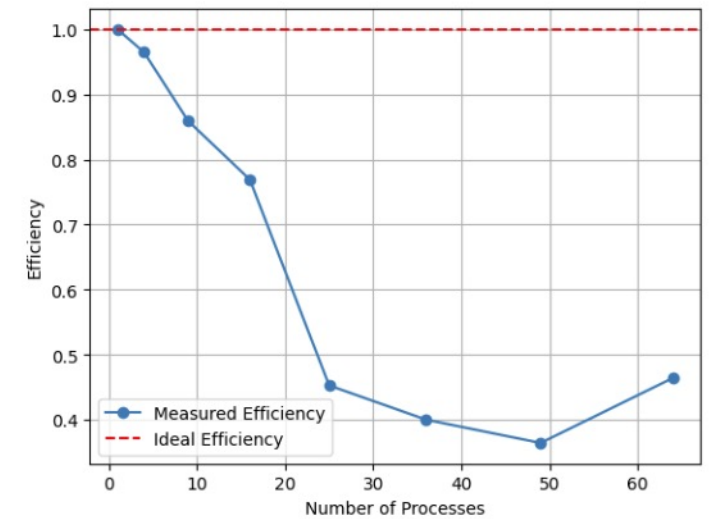
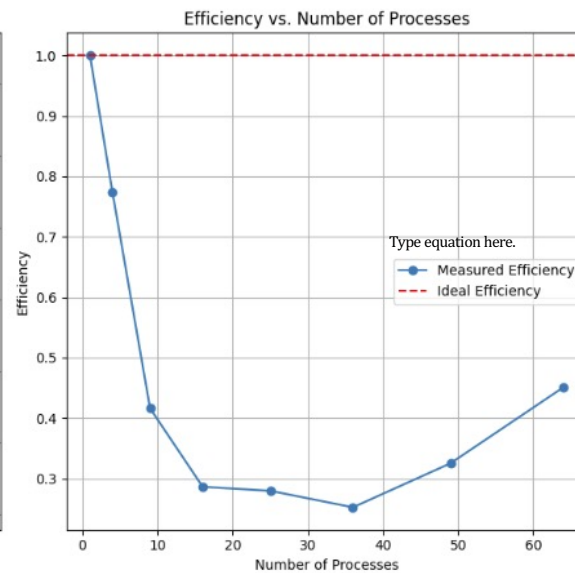


Step6. Vector updates: $d = -g + \beta d$

Scalability Experiments



Strong Scalability, $n = 2048$



Weak Scalability, $n = 256$ per process

Outputs

n	residual ($\ g\ _2$)
256	3.82×10^{-5}
512	4.43×10^{-3}
768	6.34×10^{-3}
1024	7.28×10^{-3}
1280	7.62×10^{-3}
1536	7.90×10^{-3}
1972	8.14×10^{-3}
2048	8.25×10^{-3}

$O(n^2)$

$n = 1024$, itr = 722, $r = 9.75e-6$