Automatic parallelization of vector parallel codes for preconditioned iterative solvers

Oleg Batrashev

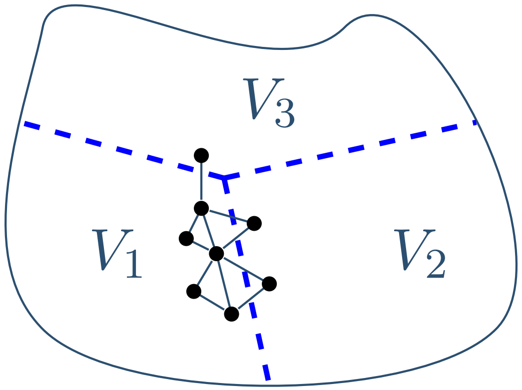
June 1, 2011

Background

- University of Tartu, Estonia (one of Baltic States)
  - Eero Vainikko: professor at Distributed Systems Group
    - Scientific Computing and HPC
  - Me: PhD student
    - HPC + Software Engineering
    - Programming Languages
- DOUG – Domain Decomposition on Unstructured Grids
  - two-level Schwarz preconditioners $M^{-1} = M_{AS}^{-1} + M_C^{-1}$
  - theory: University of Bath, UK
  - implementation: Fortran 90 + MPI

Domain and related work

Schwarz preconditioner

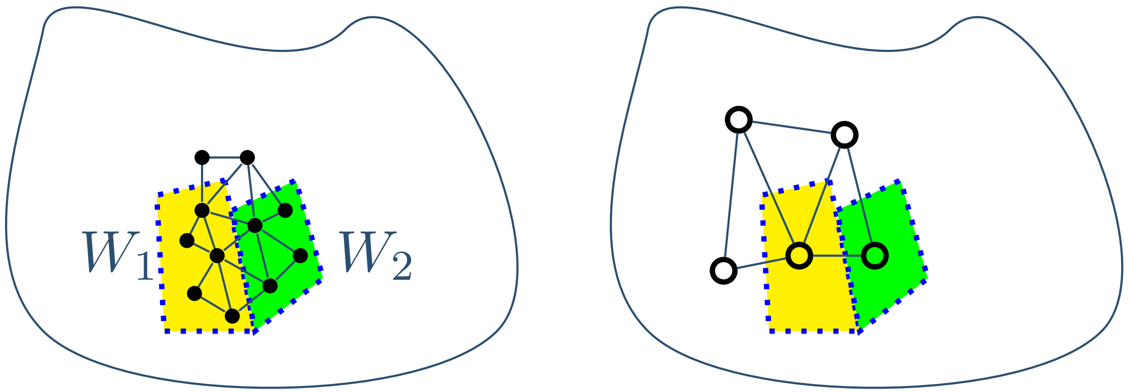


$$R_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

- subdomain *injection matrix* R_i picks the nodes (unknowns x_j) corresponding to subdomain Ω_i
- $A_i = R_i A R_i^T$ is a minor matrix of A
- During each CG iteration apply

$$M_{AS}^{-1} = \sum_{i=1}^{s} R_i^T A_i^{-1} R_i$$

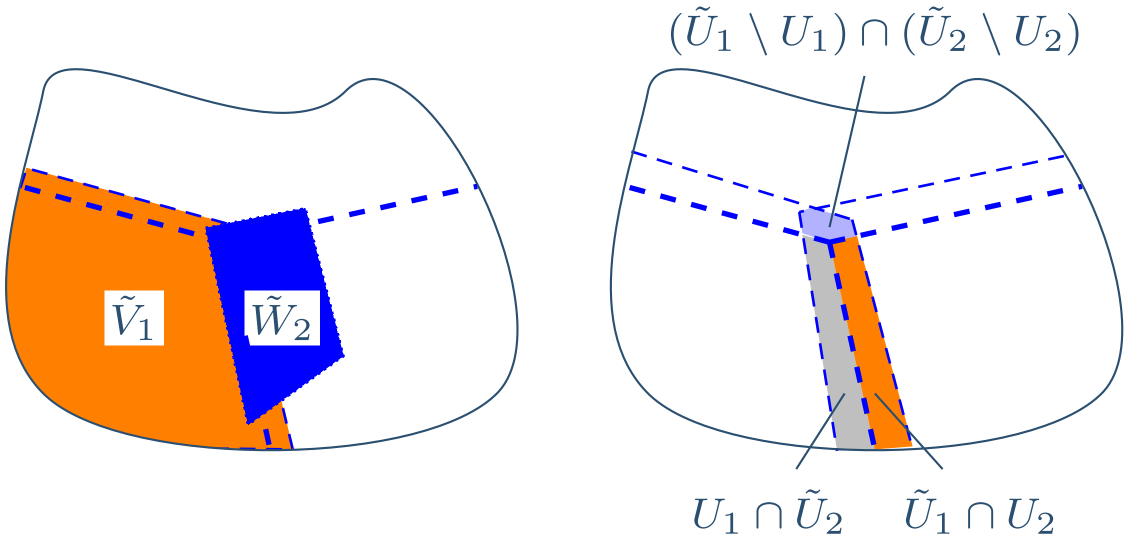
Coarse grid preconditioner



- coarse space *restriction matrix* R_c combines the nodes corresponding to supports W_i
- coarse matrix $A_c = R_c A R_c^T$ defines the problem on coarse grid
- During each CG iteration apply

$$M_C^{-1} = R_c^T A_c^{-1} R_c$$

Overlaps



- Both subdomains V_i and supports W_j may be extended
- Values on the overlap usually added (depends on algorithm)
- Process regions U_k union of all local $\tilde{V}_i$ and $\tilde{W}_j$

Patterns and problems

- Unstructured grids
  - irregular problem
  - no stencils for regular grids
- Managing overlaps on process boundaries
  - synchronize values
  - exclude duplicates:
    - dot product
    - in A_c
  - several slightly different overlaps
  - more sophisticated preconditioners

Related work

- DUNE – the Distributed and Unified Numerics Environment
  - partition value types: *interior*, *border*, *overlap*, *front*, *ghost*
  - index sets (*owner*, *ghost*)
- HPF-2, Vienna Fortran, Fortran D
  - `SPARSE(CRS(Data,Col,Row))`
  - `DECOMPOSITION, ALIGN, DISTRIBUTE`
- Nested Data Parallelism: NESL, Intel ArBB
  - array languages
  - combining scatter (histogram reduction)
  - for SMP systems

SMVM in Intel ArBB


```
void Ax(const Matrix &A,
        const dense<f64> &x,
        dense<f64> &y)
{
  dense<f64> colvals = gather(x, A.cols);
  dense<f64> mvals = colvals * A.vals;
  nested<f64> nmvals =
     reshape_nested_offsets(mvals, A.nrows);
  y = add_reduce(nmvals);
}
```

- Enough to express CG
  - PCG requires more abstractions

Our approach

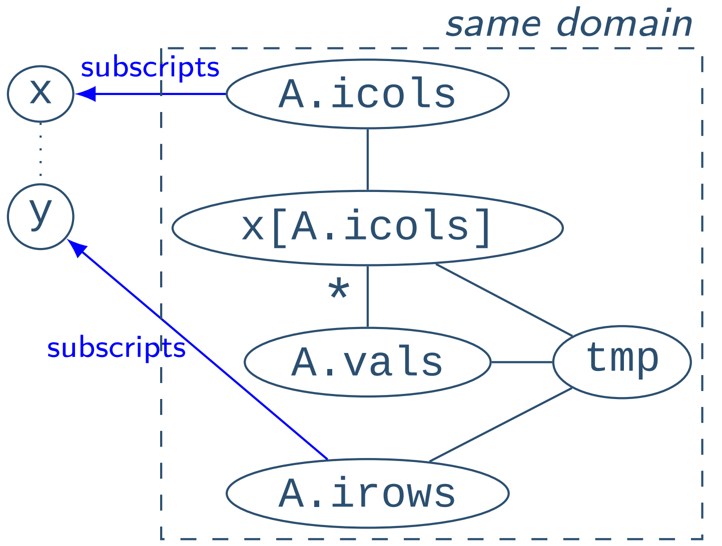
Array operations

The following is enough for Conjugate Gradient. Let `x`, `y`, and `z` are arrays:

- array creation: `y=zeros(N)`, `y=copy_like(x)`
- array copy: `y=copy(x)`
- binary, element-wise: `z=x+y`, `z=x*y`, `y=sqrt(x)`, `x==y`
- reduction: `r=reduce(x, op)`
- gather: `z=x[y]`
- scatter: `z[y]=x`
- combining scatter: `z=hreduce(y, x, op='+')`, i.e. `z[y]+=x`

Array relations


```
def Ax(A, x):
    tmp = x[A.icols]*A.vals
    y = ops.hreduce(A.irows, tmp, like=x)
    return y
```



- `A.irows` to calculate distribution
- `A.icols` to calculate ghost values

Complex array operations

The following is almost enough for 1-level Schwarz preconditioner:

- `z=index(x)` – gather indexes into array `z` of boolean array `x`
- `z=set_in(x,y)` – find `x` elements which values are in array `y`
- `z=set_union(x,y)` – combine arrays as sets
- `z=inverse(x)` – inverse array

```
def add_layer(domain, A):
    r = ops.set_in(A.irows, domain)
    t = ops.index(r)
    v = A.icols[t]
    newDomain = ops.set_union(domain, v)
    return newDomain
```

Apply on a subdomain

- Apply to a subdomain

```
def apply_prec(self, r):
    N_ITER=8
    x = ops.zeros_like(r)
    rl = r[self.d]
    xl = stationary.sym_gauss_seidel(self.Al,
      rl, N_ITER)
    x[self.d] += xl
    return x
```

- Problem: some code impossible to vectorize

Overview of analysis

- Python code with calls to *ops* package
  - special comment *"parallelize : x - domains"*
- Get python AST (Abstract Syntax Tree)
  - *ast* package starting from Python 2.6
- Generate IR (Intermediate Representation) from AST
- Analyze IR
  - Find arrays and their relations
  - Decide where to insert communication code
- Generate Python code from IR

Intermediate Representation

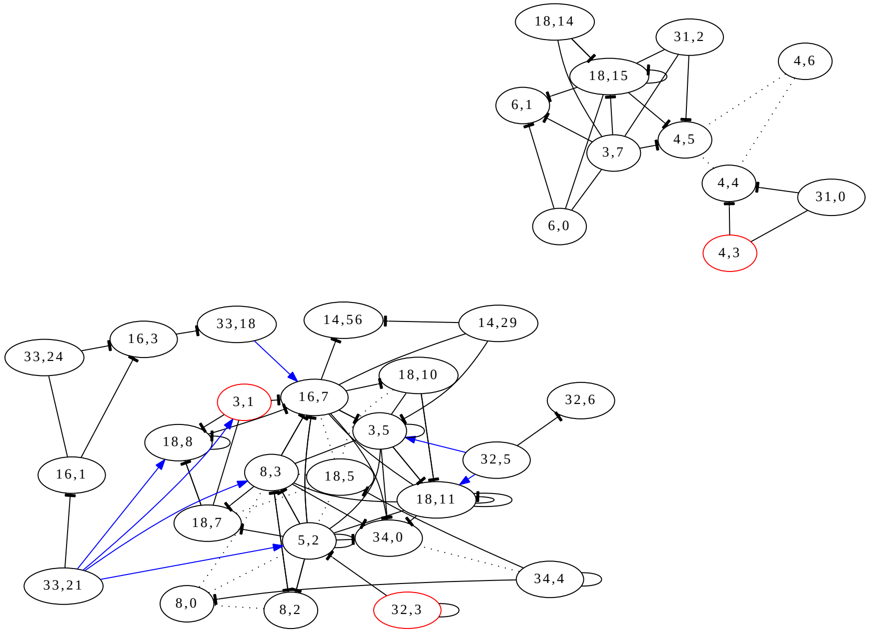

```
def Ax(A, x):
    tmp = x[A.icols]*A.vals
    y = ops.hreduce(A.irows, tmp, like=x)
    return y
```

- Corresponding IR

```
  0   = A.icols        : A(INT)
  1  := x[0]           : A(FLOAT)
  2   = A.vals         : A(FLOAT)
  3  := 1 * 2          : A(FLOAT)
 tmp  = 3              : A(FLOAT)
  5   = A.irows        : A(INT)
  6   = ops.hreduce    : oF(hreduce)
  7   = 6(5,tmp,x)     : A(FLOAT)
  y   = 7              : A(FLOAT)
  return = y           : A(FLOAT)
```

Data-flow analysis

- Pointer analysis (+ Type Analysis)
  - Find definitions: `z=ops.zeros()`, `z=x+y`
  - track each definition
- Find definition (array) relations



Distribution propagation

Using definition relation graph

- decide which belong to the same distribution
  - x=y+z, x=y[z]
- find the partitioning that has been specified
  - decide how to infer other distributions
  - ghost values

The rest: generate code

Summary

- CG and preconditioners – a lot of work
- express using vector parallel (array) code
- find array relations
  - one array defines distribution of another
  - find ghost values
- Up-to-date
  - parallel CG works
  - parallel Schwarz preconditioner is ongoing