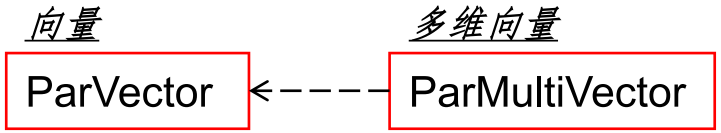
向量



Struct ParMultiVector

Constructors

ParMultiVector(MPI_Comm comm);
// 通信域comm上的空向量

ParMultiVector(MPI_Comm, int num, int size);
// 通信域comm上num个长度为size的向量

ParMultiVector(const ParMultiVector & X);
// 拷贝构造

Operator overloading

const ParVector operator()(int j) const;
// 访问第j个向量

ParMultiVector & operator=(const ParMultiVector & X) const;
// 拷贝赋值

Functions

void Free();
// 释放内存

void Allocate(int num, int size);
// 分配num个长度为size的向量

void Refer(const ParMultiVector &);
// 引用X

Struct ParVector

Constructors

ParVector(MPI_Comm comm);
// 通信域comm上的空向量

ParVector(MPI_Comm comm, int size);
// 通信域comm上长度为size的向量

ParVector(const ParVector & x);
// 拷贝构造

Operator overloading

double& operator[](int i) const;
// 访问第i个元素

ParVector & operator=(const ParVector & x);
// 拷贝赋值

Functions

void Free();
// 释放内存

void Resize(int size);
// 改变长度

void Refer(const ParVector & x);
// 引用x

void Fill(double a) const;
// 标量a填充

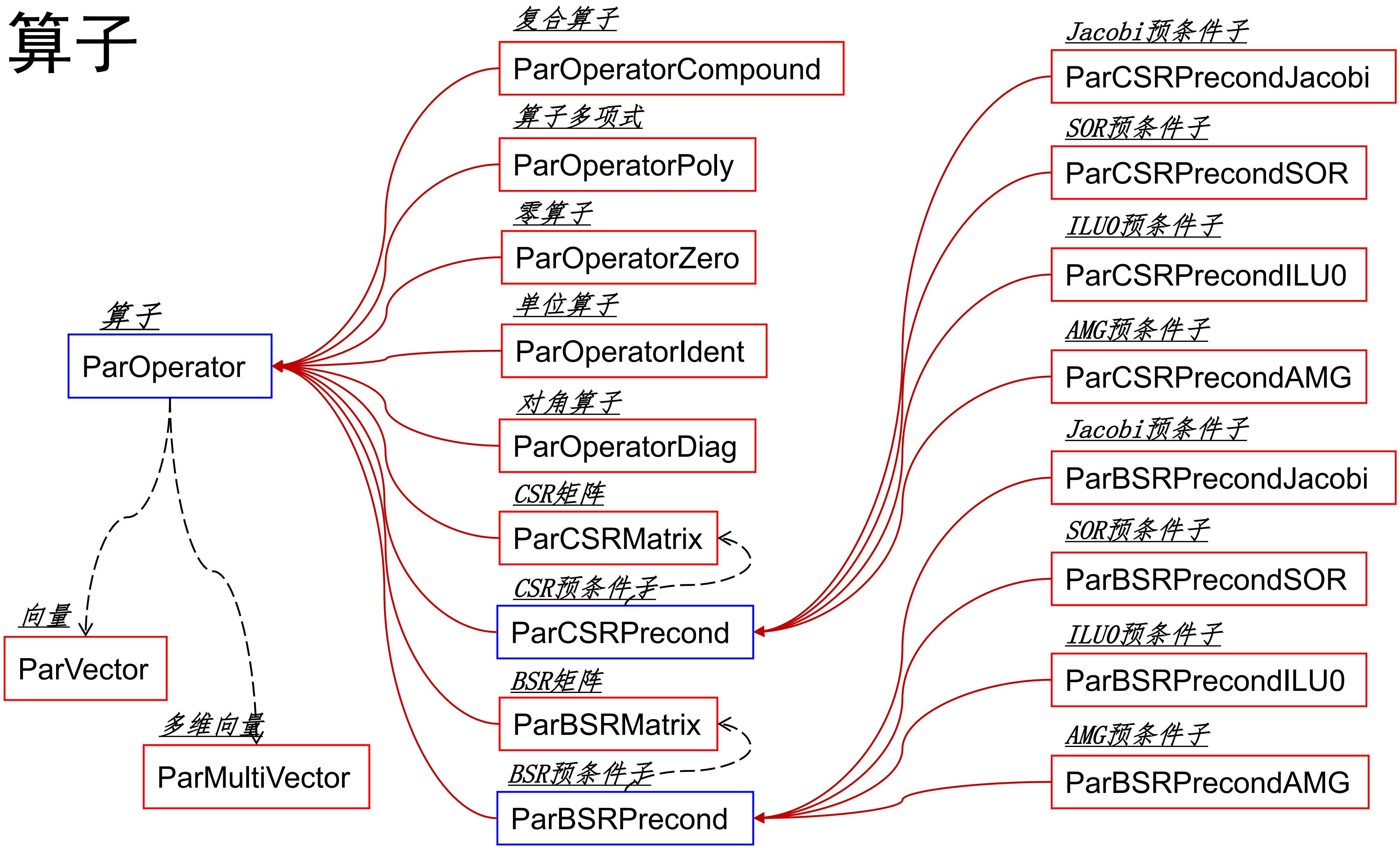
void FillRandom() const;
// 随机数填充

void Copy(const ParVector & x) const;
// 拷贝元素

void Scale(double a) const;
// this *= a

void AddScaled(double, const ParVector & x) const;
// this += a*x

算子



算子
ParOperator

向量
ParVector

多维向量
ParMultiVector

复合算子
ParOperatorCompound

算子多项式
ParOperatorPoly

零算子
ParOperatorZero

单位算子
ParOperatorIdent

对角算子
ParOperatorDiag

CSR矩阵
ParCSRMatrix

CSR预条件子
ParCSRPrecond

BSR矩阵
ParBSRMatrix

BSR预条件子
ParBSRPrecond

Jacobi预条件子
ParCSRPrecondJacobi

SOR预条件子
ParCSRPrecondSOR

ILU0预条件子
ParCSRPrecondILU0

AMG预条件子
ParCSRPrecondAMG

Jacobi预条件子
ParBSRPrecondJacobi

SOR预条件子
ParBSRPrecondSOR

ILU0预条件子
ParBSRPrecondILU0

AMG预条件子
ParBSRPrecondAMG

class ParOperator

Functions

int InSize() const; // 输入长度

int OutSize() const; // 输出长度

void Apply(const ParVector & x, const ParVector & y) const; // y = this * x

void Apply(const ParMultiVector & X, const ParMultiVector & Y) const; // Y = this * X

class ParOperatorDiag : public ParOperator

Constructors

ParOperatorDiag(const ParVector & x); // diag(x)

class ParOperatorIdent : public ParOperator

Constructors

ParOperatorIdent(MPI_Comm comm, int n); // 通信域comm上的单位矩阵 I_{n*n}

class ParCSRPrecond : public ParOperator

Functions

void Setup(const ParCSRMatrix & A) const; // Setup

class ParOperatorZero : public ParOperator

Constructors

ParOperatorIdent(MPI_Comm comm, int n); // 通信域comm上的零矩阵 O_{n*n}

class ParBSRPrecond : public ParOperator

Functions

void Setup(const ParBSRMatrix & A) const; // Setup

class ParOperatorCompound : public ParOperator

Constructors

ParOperatorCompound(double alpha, const ParOperator & A, const ParOperator & B, double beta, const ParOperator & C);

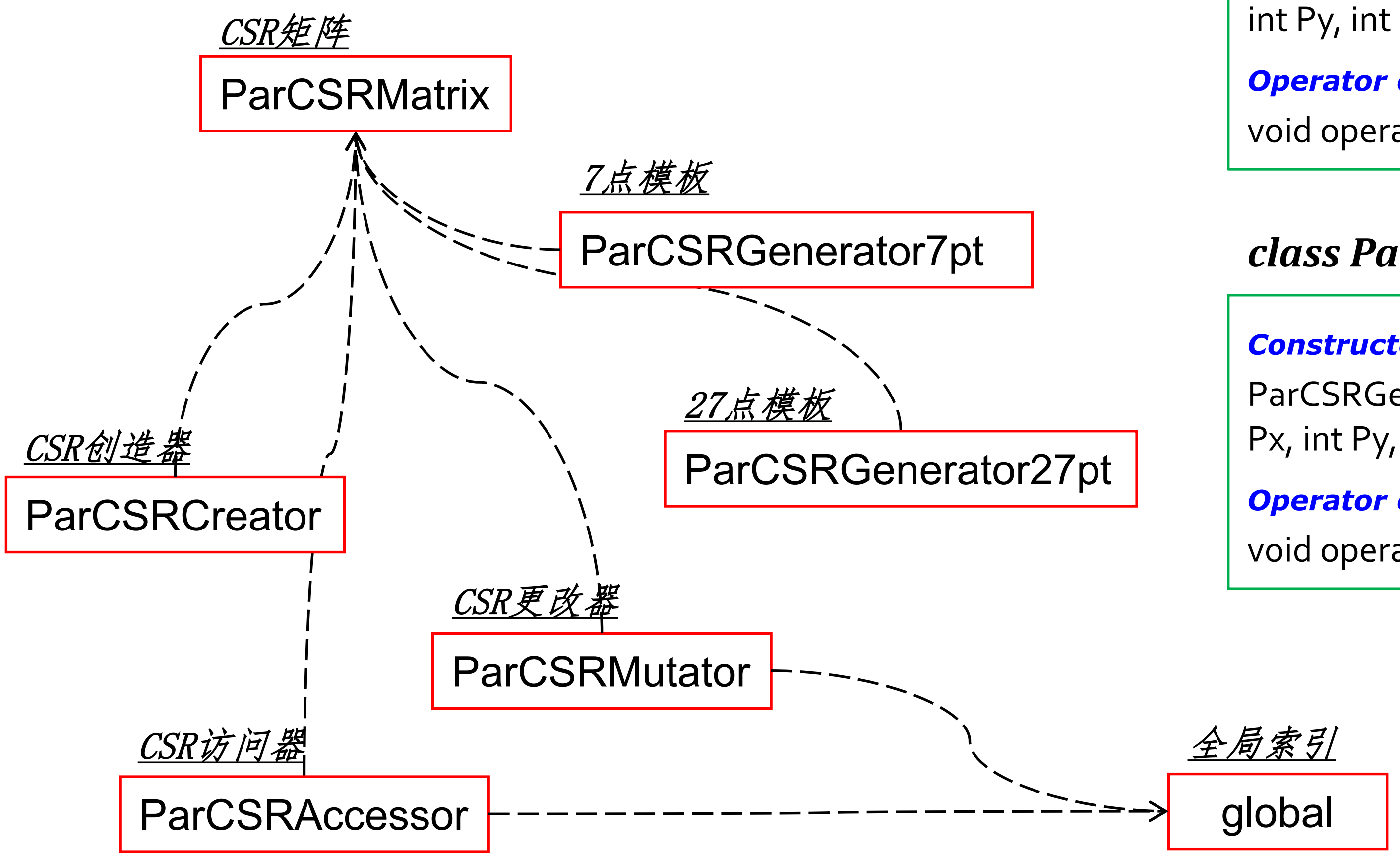
// 复合算子 alpha*A *B + beta * C

class ParOperatorPloy : public ParOperator

Constructors

ParOperatorCompound(int r, const double* a, const ParOperator & A); // 算子多项式 $a[r]* A^r + a[r-1]*A^{r-1} + ... + a[1] * A + a[0]$

矩阵生成



class ParCSRGenerator7pt

Constructor

ParCSRGenerator7pt(MPI_Comm comm, int nx, in ny, int nz, int Px, int Py, int Pz); // 通信域、局部规模和进程拓扑

Operator overloading

void operator()(const ParCSRMatrix & A) const; // 生成矩阵

class ParCSRGenerator27pt

Constructor

ParCSRGenerator27pt(MPI_Comm comm, int nx, in ny, int nz, int Px, int Py, int Pz); // 通信域、局部规模和进程拓扑

Operator overloading

void operator()(const ParCSRMatrix & A) const; // 生成矩阵

class ParCSRCreator

Constructor

```
ParCSRCreator(MPI_Comm comm, int rows, int cols);    // 通信域和维度
```

Operator overloading

```
void operator()(const ParCSRMatrix & A) const;    // 生成空矩阵
```

class ParCSRMutator

Constructor

```
ParCSRMutator(const ParCSRMatrix & A);    // 初始矩阵
```

Operator overloading

```
void operator()(const ParCSRMatrix & A);    // 生成更改矩阵
```

Functions

```
double * Find(int row, global col) const;      // 查找非零元(row, col)
double * Insert(int row, global col) const;    // 插入非零元(row, col)
void Erase(int row, global col) const;         // 擦除非零元(row, col)
```

class ParCSRAccessor

Constructor

```
ParCSRAccessor(const ParCSRMatrix & A);    // 访问矩阵A
```

Functions

```
double * Find(int row, global col) const;    //查找非零元(row, col)
```

struct ParCSRMatrix : public ParOperator

Constructor

```
ParCSRMatrix(const ParCSRMatrix & A);
                                          // 拷贝构造
```

Operator overloading

```
ParCSRMatrix & operator=(const ParCSRMatrix & A);
                                          // 拷贝赋值
```

Functions

```
void Free();
                                          // 释放内存
void Refer(const ParCSRMatrix & A);
                                          // 引用A
void ExchangeHalo(const ParVector & x) const;
                                          // 交换环
void SetupHalo();
                                          // Setup环
```

struct global

Constructor

```
global(int indl, int from);    // 局部索引和进程ID
```

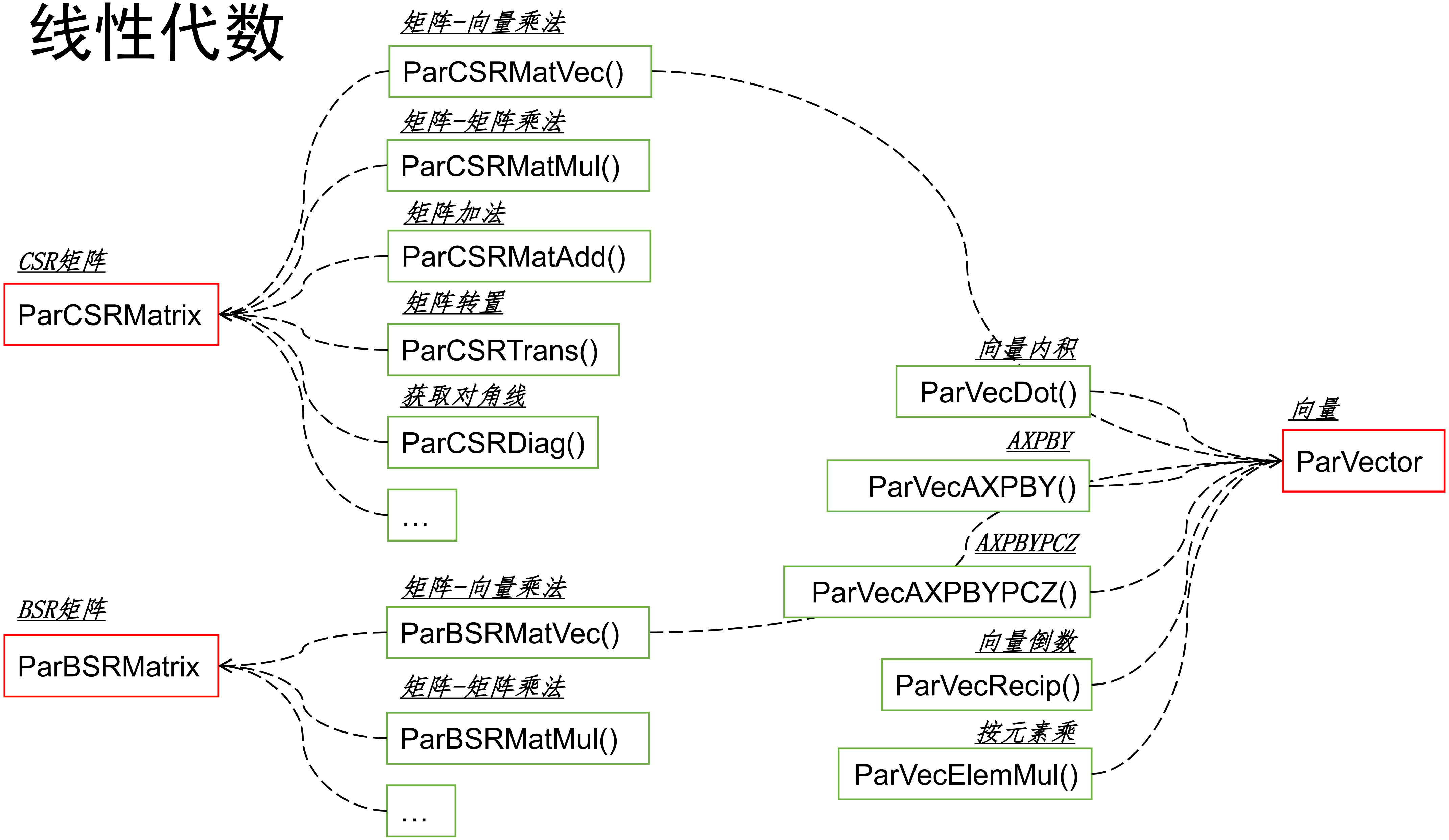
Operator overloading

```
bool operator<(const global& a) const;    // 比较函数
```

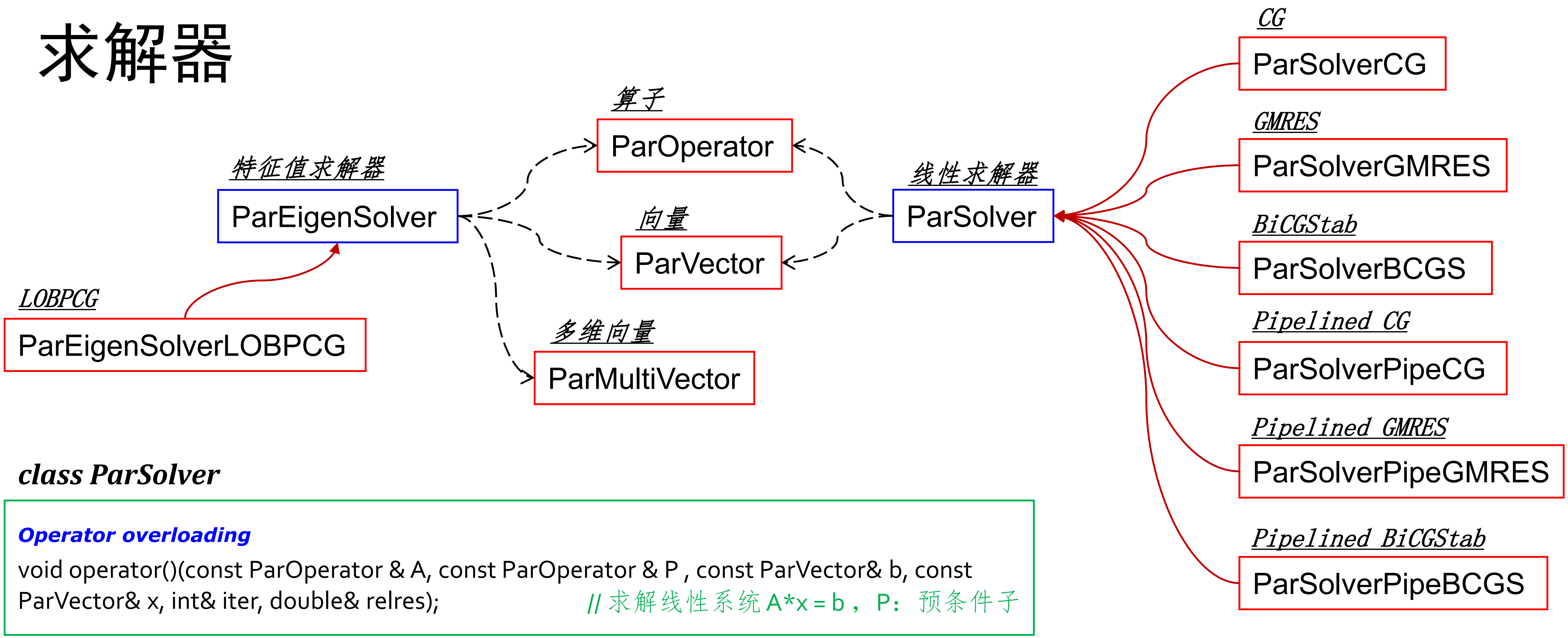
Functions

```
int local() const;     //局部索引
int owner() const;     //进程ID
```

线性代数


矩阵-向量乘法
ParCSRMatVec()
矩阵-矩阵乘法
ParCSRMatMul()
矩阵加法
ParCSRMatAdd()
矩阵转置
ParCSRTrans()
获取对角线
ParCSRDiag()
…
CSR矩阵
ParCSRMatrix
BSR矩阵
ParBSRMatrix
矩阵-向量乘法
ParBSRMatVec()
矩阵-矩阵乘法
ParBSRMatMul()
…
向量内积
ParVecDot()
AXPBY
ParVecAXPBY()
AXPBYPCZ
ParVecAXPBYPCZ()
向量倒数
ParVecRecip()
按元素乘
ParVecElemMul()
向量
ParVector

求解器



算子

ParOperator

特征值求解器

ParEigenSolver

向量

ParVector

线性求解器

ParSolver

LOBPCG

ParEigenSolverLOBPCG

多维向量

ParMultiVector

CG

ParSolverCG

GMRES

ParSolverGMRES

BiCGStab

ParSolverBCGS

Pipelined CG

ParSolverPipeCG

Pipelined GMRES

ParSolverPipeGMRES

Pipelined BiCGStab

ParSolverPipeBCGS

class ParSolver

Operator overloading

```
void operator()(const ParOperator & A, const ParOperator & P , const ParVector& b, const
ParVector& x, int& iter, double& relres);            // 求解线性系统 A*x = b ， P： 预条件子
```

class ParEigenSolver

Operator overloading

```
void operator()(const ParOperator & A, const ParOperator & B, const ParOperator & T, const ParMultiVector & Y, const ParMultiVector & X,
const Vector & Lambda, int & iter, const Vector & Res) const;          // 求解特征值 A*X = B* X * Lambda， T： 预条件子， Y： 约束 （Y^T*X = 0）
```

算法

Subspace Iterative Methods	CG	Krylov Subspace Methods
	BiCGStab	
	GMRES	
	PipeCG	Pipelined, Hiding Global Reduction
	PipeBiCGStab	
	PipeGMRES	
	LOBPCG	Generalized Symmetric EigenSolver

Preconditioner	AMG	HMIS, PMIS and Aggressive Coarsening;
		Long Range Interpolation; Smoothed Aggregation;
		Jacobi, SOR, ILU and Chebyshev Smoother

	Jacobi	
	SOR	SOR, SSOR
	ILU	ILU(0)

示例：分布式线性系统求解 mpirun -n 8 ./main

```
ParCSRMatrix A{MPI_COMM_WORLD};                                      // 预定义
ParVector x{MPI_COMM_WORLD}, b{MPI_COMM_WORLD};
ParCSRPrecondAMG P;

int nx = 128, ny = 128, nz = 128;                                    // 设置问题规模
int Px = 2, Py = 2, Pz = 2;                                          // 设置进程拓扑（进程数 = Px*Py*Pz）
ParCSRGenerator27pt{MPI_COMM_WORLD, nx, ny, nz, Px, Py, Pz}(A);      // 生成27点矩阵
A.SetupHalo();                                                       // Setup交换环

x.Resize(128*128*128); x.Fill(0.0);                                  // 生成初始解向量
b.Resize(128*128*128); b.FillRandom();                               // 生成右端向量

P. CoarsenType = 0;                                                  // 配置AMG参数
P. InterpType = 1;                                                   // …
P. SmoothType = 2;
…
P.Setup(A);                                                          // Setup AMG

int iter; double relres;
ParSolverCG{100, 1.0e-08}(A, P, b, x, iter, relres);                 // 用AMG-CG方法求解 Ax = b
```

示例：特征值问题求解 mpirun -n 8 ./main

```
ParCSRMatrix A{MPI_COMM_WORLD};                                         // 预定义
ParMultiVector X{MPI_COMM_WORLD};
ParCSRPrecondILU0 P;

int nx = 128, ny = 128, nz = 128;                                       // 设置问题规模
int Px = 2, Py = 2, Pz = 2;                                             // 设置进程拓扑（进程数 = Px*Py*Pz）
ParCSRGenerator27pt{MPI_COMM_WORLD, nx, ny, nz, Px, Py, Pz}(A);         // 生成27点矩阵
A.SetupHalo();                                                          // Setup交换环

X.Allocate(4, 128*128*128);
for (int i = 0; i < 4; ++i)
    X(i).FillRandom();                                                  // 生成初始解向量

P.Setup(A);                                                             // Setup ILU

int iter; Vector Lambda, Res;
Lambda.Resize(4); Res.Resize(4);
ParSolverLOBPCG{100, 1.0e-08}(A, P, X, Lambda, iter, Res);              // 用ILU0-LOBPCG方法求解特征值问题 AX = X*Lambda
```