ulm university universität ulm

A Reimplementation of the Finite Element Method Software Package using FLENS

Mazen Ali, Christopher Davis
University of Ulm
August 5, 2013

Introduction

Introduction

Introduction

- Primary aim: to reimplement the basic FEM package using the Flexible Library for Efficient Numerical Solutions (FLENS).
  - ↪ FLENS offers implementations of matrix/vector storage types, together with the associated linear algebra operations.

Introduction

- Primary aim: to reimplement the basic FEM package using the Flexible Library for Efficient Numerical Solutions (FLENS).
  - ↪ FLENS offers implementations of matrix/vector storage types, together with the associated linear algebra operations.

- Objectives:

Introduction

- Primary aim: to reimplement the basic FEM package using the Flexible Library for Efficient Numerical Solutions (FLENS).
  - ↪ FLENS offers implementations of matrix/vector storage types, together with the associated linear algebra operations.

- Objectives:
  1. Replace all data storage objects in the FEM package with FLENS-based objects.

Introduction

- Primary aim: to reimplement the basic FEM package using the Flexible Library for Efficient Numerical Solutions (FLENS).
  - ↪ FLENS offers implementations of matrix/vector storage types, together with the associated linear algebra operations.

- Objectives:
  1. Replace all data storage objects in the FEM package with FLENS-based objects.
  2. Replace linear algebra operations with BLAS equivalents.

Introduction

- Primary aim: to reimplement the basic FEM package using the Flexible Library for Efficient Numerical Solutions (FLENS).
  - ↪ FLENS offers implementations of matrix/vector storage types, together with the associated linear algebra operations.

- Objectives:
  1. Replace all data storage objects in the FEM package with FLENS-based objects.
  2. Replace linear algebra operations with BLAS equivalents.
  3. Offer two versions of solvers, one using BLAS notation, one using overloaded operators.

Replacing data storage objects

Replacing data storage objects

Replacing data storage objects

- Some objects can be directly converted:

$$\begin{aligned} \texttt{Vector} &\rightarrow \texttt{DenseVector}\langle\texttt{Array}\langle\texttt{double}\rangle\ \rangle \\ \texttt{IndexVector} &\rightarrow \texttt{DenseVector}\langle\texttt{Array}\langle\texttt{int}\rangle\ \rangle \end{aligned}$$

Replacing data storage objects

- Some objects can be directly converted:

$$\begin{aligned} \texttt{Vector} &\rightarrow \texttt{DenseVector}\langle\texttt{Array}\langle\texttt{double}\rangle\ \rangle \\ \texttt{IndexVector} &\rightarrow \texttt{DenseVector}\langle\texttt{Array}\langle\texttt{int}\rangle\ \rangle \end{aligned}$$

- FLENS offers a CRS matrix implementation, but it must be initialised from a sparse matrix using the coordinate storage scheme.

The FLENSDataVector

The FLENSDataVector

- The biggest change was the `DataVector` storage class, since it contains additional objects and methods for the parallel MPI communications.

The FLENSDataVector

- The biggest change was the `DataVector` storage class, since it contains additional objects and methods for the parallel MPI communications.
- We replaced this class with our own `FLENSDataVector` class:
  ↪ Derived from a FLENS `DenseVector`:

```
1 template <typename VTYPE = FLvNonMPI>
2 struct FLENSDataVector
3          : public DenseVector<Array<double>
4 { ... }
```

The FLENSDataVector

- The biggest change was the `DataVector` storage class, since it contains additional objects and methods for the parallel MPI communications.
- We replaced this class with our own `FLENSDataVector` class:
  - ↪ Derived from a FLENS `DenseVector`:

```
1 template <typename VTYPE = FLvNonMPI>
2 struct FLENSDataVector
3          : public DenseVector<Array<double>
4 { ... }
```

  - ↪ Contains almost all objects and methods from the `DataVector`:

```
1 const Coupling &coupling;
2
3 void typeII_2_I();
4 void typeI_2_II();
5
6 void commCrossPoints();
7 void commBoundaryNodes();
```

The FLENSDataVector

The FLENSDataVector

- Instead of the MPI vector type as a member enumerated object, the type is specified (permanently) at instantiation:

```
1 //Vector types:
2 class FLvNonMPI;
3 class FLvTypeI;
4 class FLvTypeII;
```

The FLENSDataVector

- Instead of the MPI vector type as a member enumerated object, the type is specified (permanently) at instantiation:

```
1 //Vector types:
2 class FLvNonMPI;
3 class FLvTypeI;
4 class FLvTypeII;
```

```
1 FLENSDataVector<FLvTypeI> myVector(20, myCoupling);
```

The FLENSDataVector

- Instead of the MPI vector type as a member enumerated object, the type is specified (permanently) at instantiation:

```
1 //Vector types:
2 class FLvNonMPI;
3 class FLvTypeI;
4 class FLvTypeII;
```

```
1 FLENSDataVector<FLvTypeI> myVector(20, myCoupling);
```

- Clever specialisations of constructors prevents inappropriate usage.

The FLENSDataVector

- Instead of the MPI vector type as a member enumerated object, the type is specified (permanently) at instantiation:

```
1 //Vector types:
2 class FLvNonMPI;
3 class FLvTypeI;
4 class FLvTypeII;
```

```
1 FLENSDataVector<FLvTypeI> myVector(20, myCoupling);
```

- Clever specialisations of constructors prevents inappropriate usage.

  ↪ More rigorous.

The FLENSDataVector

- Instead of the MPI vector type as a member enumerated object, the type is specified (permanently) at instantiation:

```
1 //Vector types:
2 class FLvNonMPI;
3 class FLvTypeI;
4 class FLvTypeII;
```

```
1 FLENSDataVector<FLvTypeI> myVector(20, myCoupling);
```

- Clever specialisations of constructors prevents inappropriate usage.
  - ↪ More rigorous.
  - ↪ Makes many assertions redundant, as all type checking is done at compile time.

BLAS and the Conjugate Gradient solver

BLAS

BLAS

- Basic Linear Algebra Suite
  - ↪ Included in FLENS.
  - ↪ Available via overloaded operators in FLENS.

BLAS

- Basic Linear Algebra Suite
  - ↪ Included in FLENS.
  - ↪ Available via overloaded operators in FLENS.
- GotoBLAS / OpenBLAS / ATLAS
  - ↪ Basically 'supercharged' BLAS routines, usually optimised for a specific processor.
  - ↪ Can be linked into FLENS (to replace FLENS BLAS routines) trivially (one compiler flag + linker paths).

BLAS

- FEM Package function call:

```
1 CRSmatVec(p,A,x);
2 add(r2,p,-1.);
```

BLAS

- Our function call via BLAS:

```
1 blas::mv(NoTrans, 1., A, x, 0., p);
2 blas::axpy(-1., p, r2);
```

BLAS

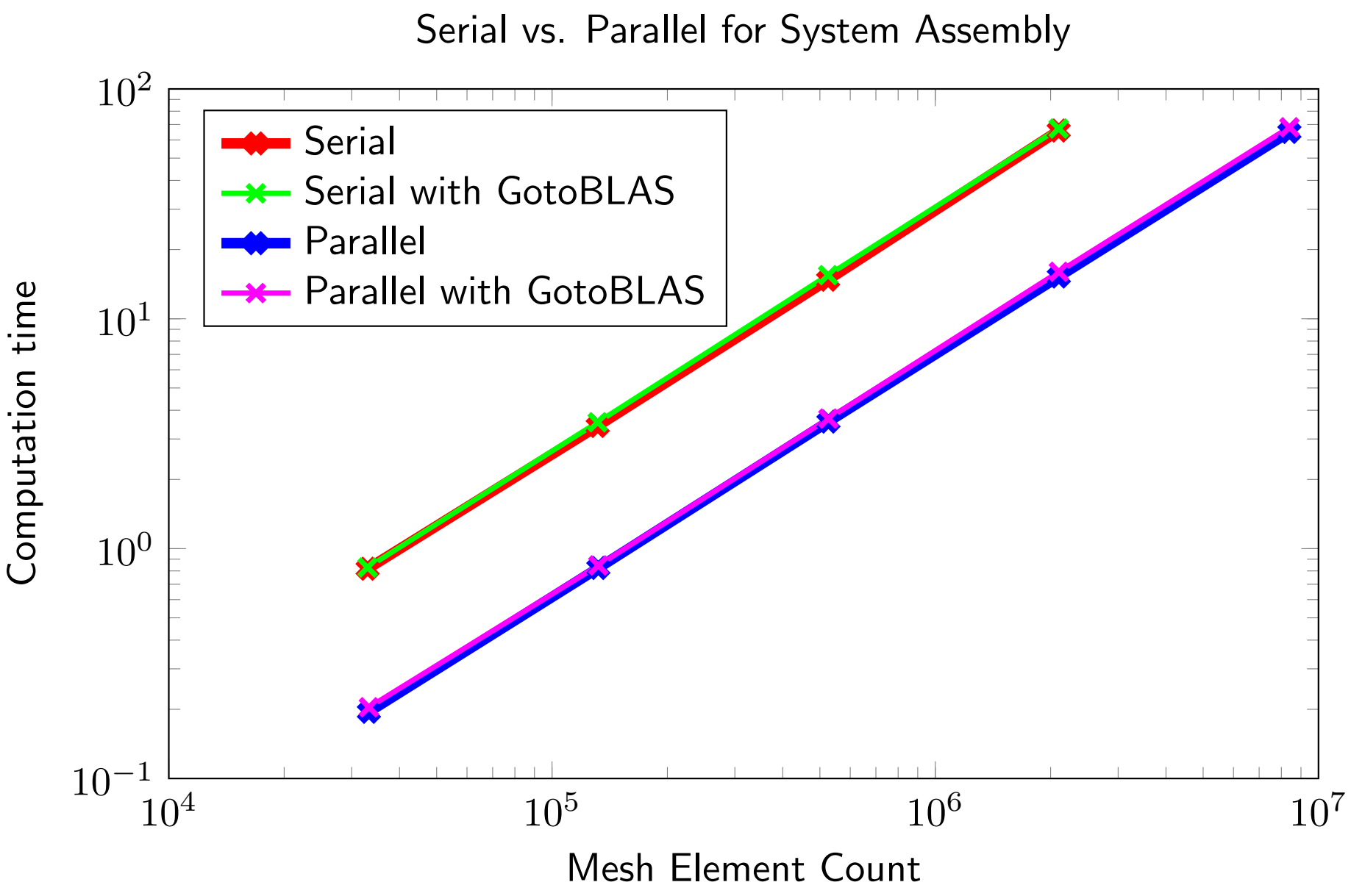
- Our function call via overloaded operators:

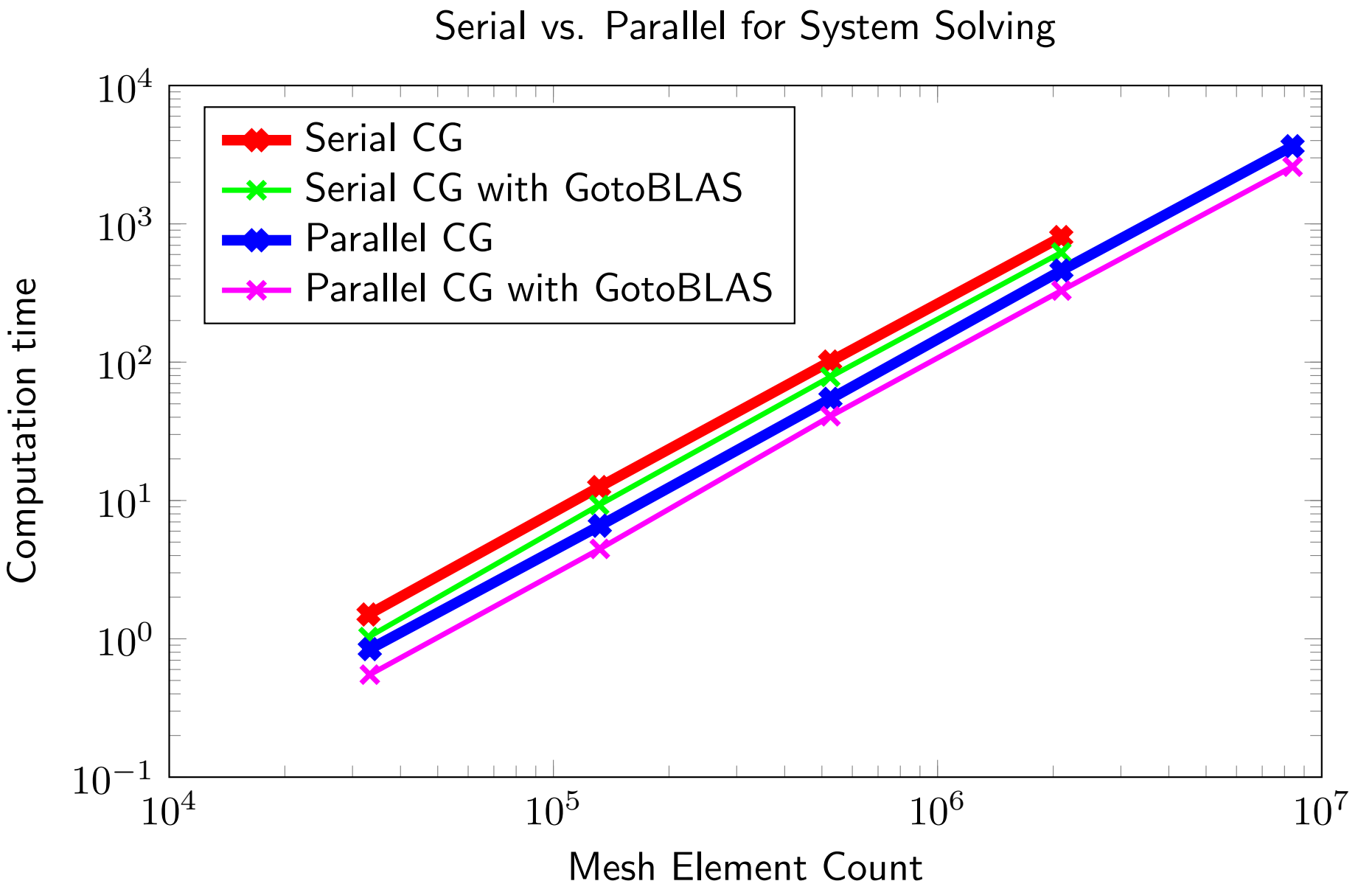
```
1 p = A*x;
2 r2 = r2 - p;
```

BLAS

- Care was taken when overloading BLAS operations that require MPI communications:

```
1  //Overloaded dot - performs appropriate communication:
2  double
3  dot(const FLENSDataVector<FLvTypeI> &x1,
4      const FLENSDataVector<FLvTypeII> &x2)
5  {
6    //Upcast to DenseVector, and use the standard blas::dot:
7    double value =
8        blas::dot(
9          *static_cast<const DenseVector<Array<double> > *>(&x1),
10         *static_cast<const DenseVector<Array<double> > *>(&x2)
11       );
12
13   double v = 0;
14   //*** Communication to add values from other processes ***/
15   MPI::COMM_WORLD.Allreduce(&value,&v,1,MPI::DOUBLE,MPI::SUM);
16
17   return v;
18 }
```


Serial vs. Parallel for System Assembly
Serial
Serial with GotoBLAS
Parallel
Parallel with GotoBLAS
Computation time
Mesh Element Count
10^2
10^1
10^0
10^{-1}
10^4
10^5
10^6
10^7


Serial vs. Parallel for System Solving
Serial CG
Serial CG with GotoBLAS
Parallel CG
Parallel CG with GotoBLAS
Computation time
Mesh Element Count
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{4}
10^{5}
10^{6}
10^{7}

The End