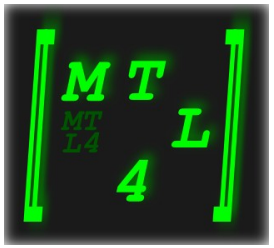
Generic Compressed Matrix Insertion

PETER GOTTSCHLING – SMARTSOFT/TUD
DAG LINDBO – KUNGLIGA TEKNISKA HÖGSKOLAN

SmartSoft – TU Dresden
Peter.Gottschling@smartsoft-computing.com
Tel.: +49 (0) 351 463 34018

TECHNISCHE UNIVERSITÄT DRESDEN

Matrix Template Library 4



- Generic library for high-performance numeric operations in mathematical notation
- Many new techniques as implicit enable-if and meta-tuning
- Most modern iterative solvers
- Focus on high-performance simulation: FEM/XFEM/FVM/FDM
- Commercial version in preparation
  - Parallel version in progress
  - Multi-core, GPU support and multigrid in near future

FEniCS

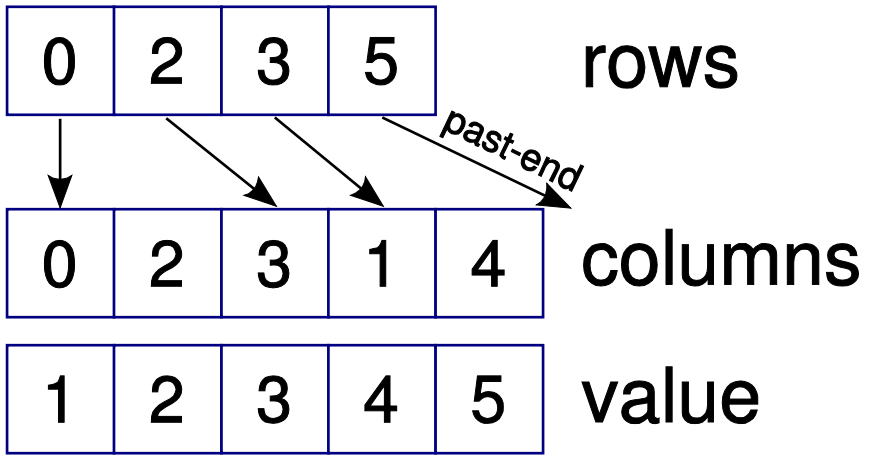
- Free software for solving differential equations
- FFC – FEniCS Form Compiler
  - High-level math language for formulating differential equations
  - Generate C++ code
- DOLFIN – generic FEM kernel
  - C++ library for FEM cores: assembler, mesh and function abstraction
  - Interface to uBLAS, PETSc, Trillinos, and MTL4
- Paper focus in matrix assembly

SmartSoft

KTH VETENSKAP OCH KONST

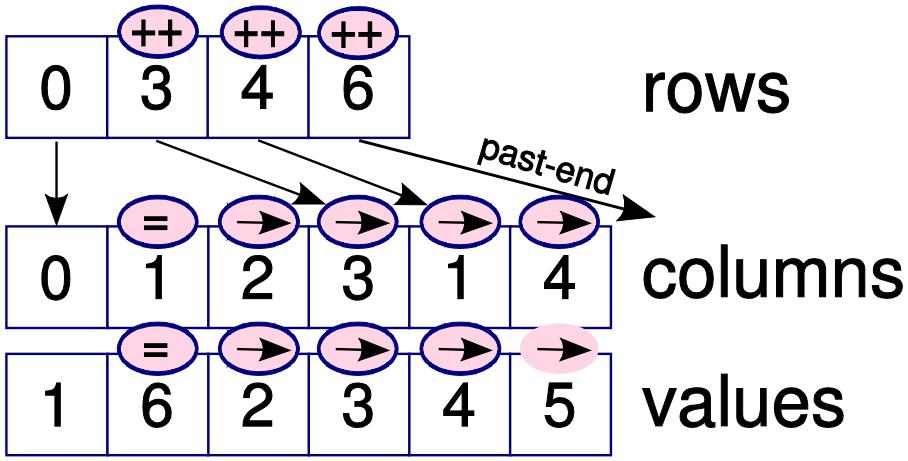
Compressed Sparse Row Format

- Most common general-purpose sparse format
- Entries sorted
- Kind of run-length encoding on rows



In-Flight Insertion

- Very simple use
  - Like dense matrices
- Simple realization
- Extremely expensive
  - All following entries are changed
  - Quadratic complexity



Two-phase Insertion

- Dedicated insertion phase
- Matrix is available after terminating insertion
- Later modification impossible
- Works for distributed matrices as well
  - Used in PETSc, includes construction of communication buffers for dist. SpMVP
  - Janus derives its name from it (two faces)

Inserter Concept in MTL4

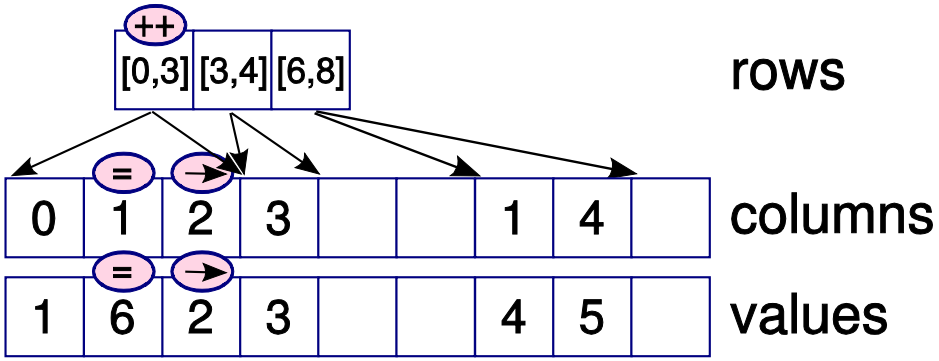
- Inserter = object providing operations to set up other objects, e.g. matrices or vectors, efficiently
- Insertion phase lasts as long as inserter lives
  - Insert within a scope (block, function)
- Matrix ready when inserter destroyed
- Later insertion possible with another inserter
- Extends to distributed matrices and vectors
- MTL4 inserters have minimal memory usage

Using Inserters

```
int main(int argc, char* argv[])
{
        compressed2D<float>   A(3, 5);
        {
           matrix::inserter<compressed2D<float> > ins(A);
           ins[0][0] << 1.0; ins[0][2] << 2.0;
           ins[1][3] << 3.0;
           ins[2][1] << 4.0; ins[2][4] << 5.0;
        }
        std::cout << "A is\n" << A << '\n';
        return 0;
}
```

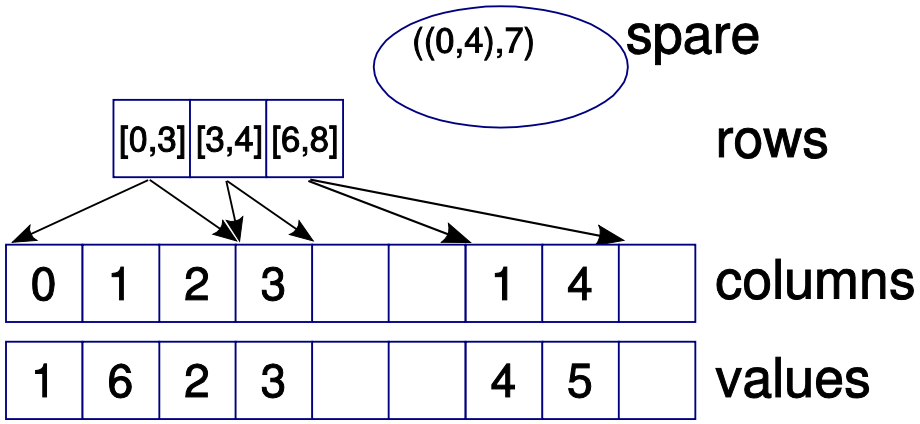
Direct Insertion

- Reserve *s* entries per row
- Find insert position
  - By linear or binary search
- Move remainder in row
- Linear in *s*
  - That is constant



Indirect Insertion

- For saturated rows use “spare” container
- std::map of index pair
- Logarithmic in number of spare entries
- Additional allocation
- About 10 times slower than direct insertion

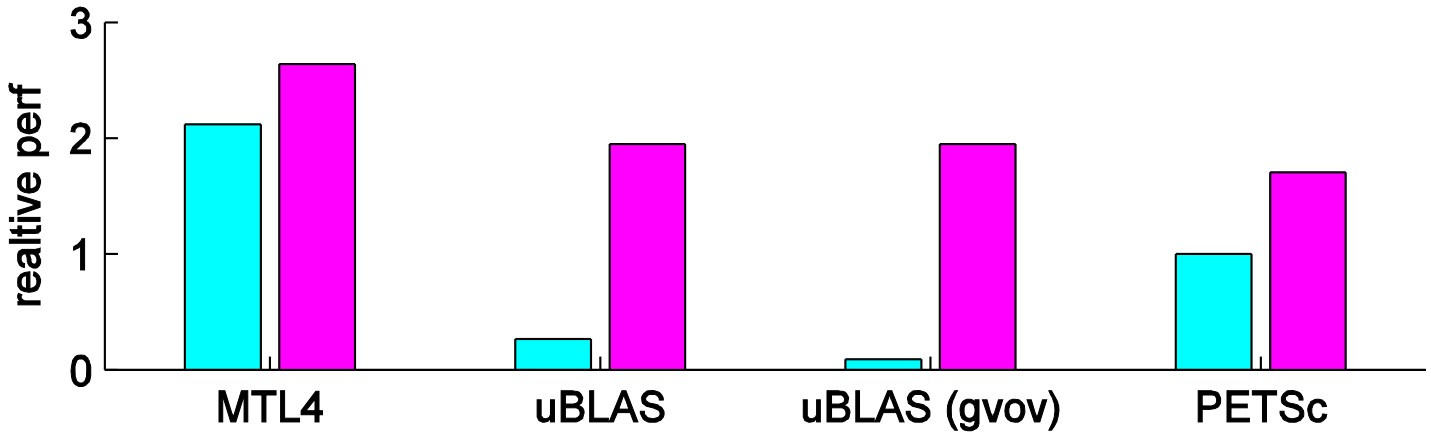


SmartSoft

KTH VETENSKAP OCH KONST

Benchmark: Assembly rate with ascending rows

- 10,000 rows, 5 non-zeros/row
- MTL4: 46 million entries per second
- uBLAS: 5.9 million entries per second
- uBLAS (gov): 2 million entries per second
- PETSc: 22 million entries per second

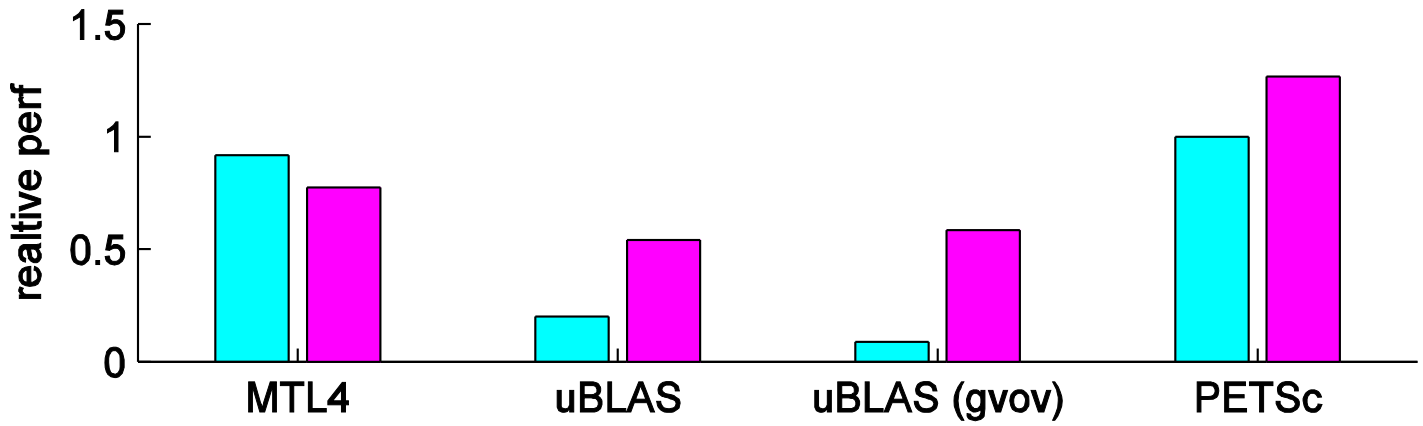


SmartSoft

KTH VETENSKAP OCH KONST

Benchmark: Assembly rate with ascending rows

- 100,000 rows, 50 non-zeros/row
- MTL4: 29.6 million entries per second
- uBLAS: 6.5 million entries per second
- uBLAS (gov): 2.8 million entries per second
- PETSc: 32.3 million entries per second

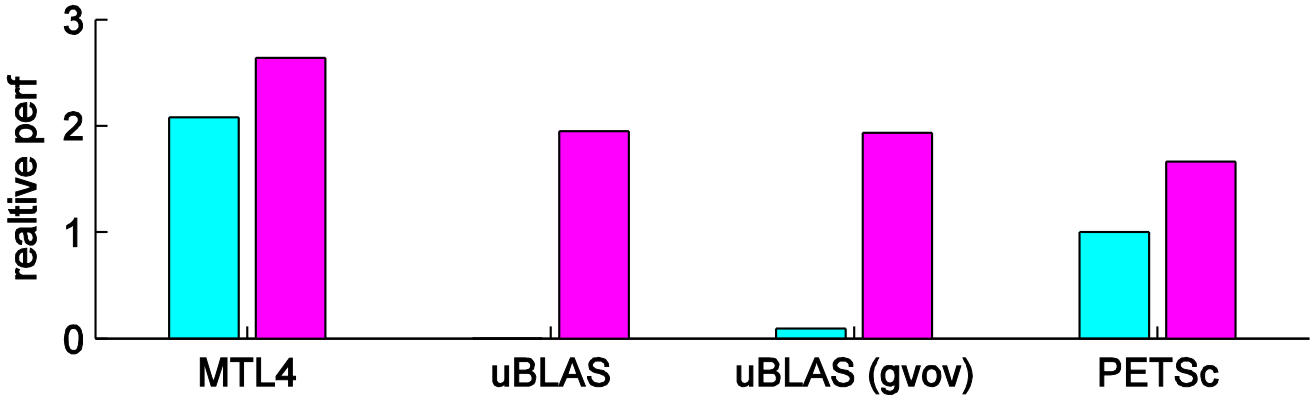


SmartSoft

KTH VETENSKAP OCH KONST

Benchmark: Assembly rate with random rows

- 10,000 rows, 5 non-zeros/row
- MTL4: 41.4 million entries per second
- uBLAS: 31,300 entries per second
- uBLAS (gov): 1.9 million entries per second
- PETSc: 19.9 million entries per second

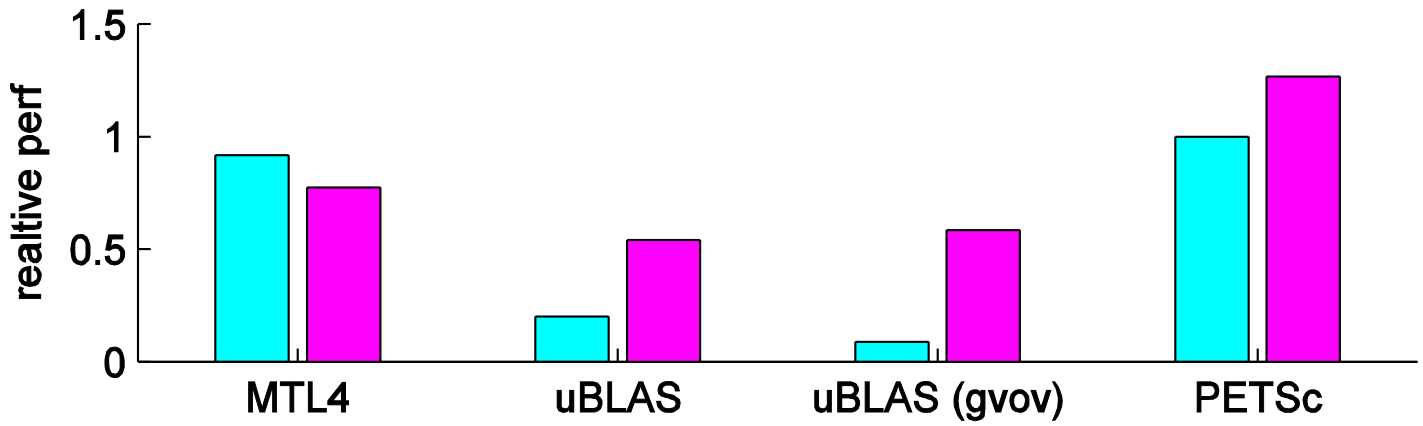


SmartSoft

KTH VETENSKAP OCH KONST

Benchmark: Assembly rate with random rows

- 100,000 rows, 50 non-zeros/row
- MTL4: 25.6 million entries per second
- uBLAS: measuring abandonned
- uBLAS (gov): 2.7 million entries per second
- PETSc: 25.6 million entries per second

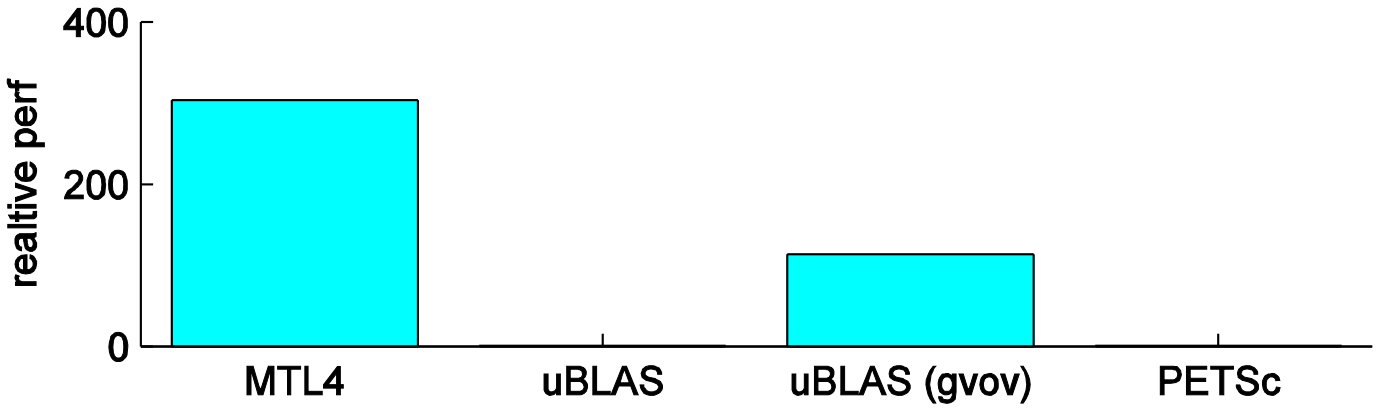


SmartSoft

KTH VETENSKAP OCH KONST

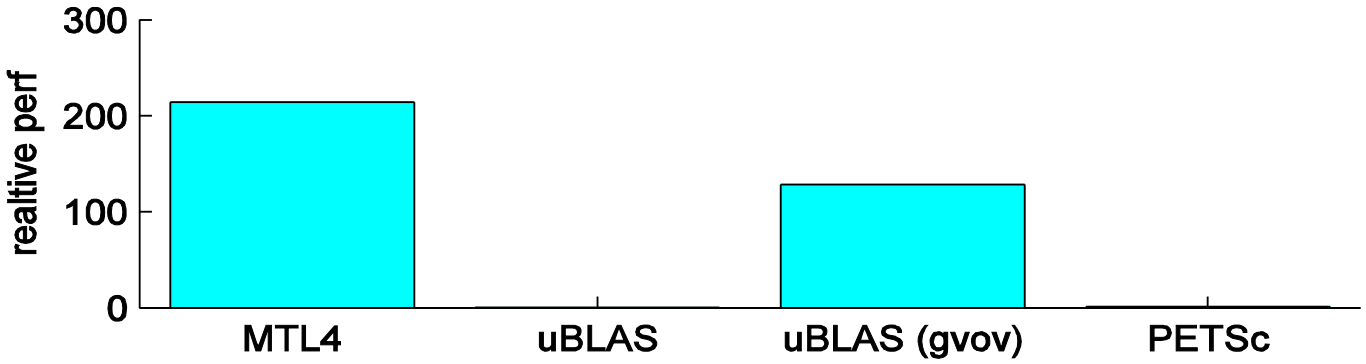
Benchmark: Assembly rate with entirely random entries

- 10,000 rows, 5 non-zeros/row
- MTL4: 4.8 million entries per second
- uBLAS: 16,700 entries per second
- uBLAS (gov): 1.8 million entries per second
- PETSc: 15,900 entries per second



Benchmark: Assembly rate with random rows

- 10,000 rows, 50 non-zeros/row
- MTL4: 2.9 million entries per second
- uBLAS: 3,340 entries per second
- uBLAS (gov): 1.7 million entries per second
- PETSc: 13,400 entries per second



SmartSoft

KTH VETENSKAP OCH KONST

Conclusions