VeloCty

An optimising static compiler for Matlab and NumPy

Sameer Jagdale
McGill University
Advisor :
Prof. Laurie Hendren

Scientific Languages

- Easy prototyping
- Gentle learning curve
- However, poor performance compared to statically compiled languages

Solution 1 : Rewrite code in C++/ Fortran

- Type and dimension declarations
- Memory management
- Increased code size
- Steeper learning curve

Solution 2: Auto compilation of whole programs to C++/Fortran

Many Functions cannot be compiled ahead of time.

```
function result = mainFunction()
    A = rand(3,3);
    X = load('dataFile.mat');
    %operations on X.
    R = X(:);
    result = coreFunction(R,A);
end;
```

VeloCty : Compile ‘hot’ code sections to C++/Fortran

- Functions that contain wild or dynamic features can be ignored.
- Users can continue programming in preferred high-level language.
- Used to generate a personalised library of commonly required functions.

Agenda

- Background
- Introduction
- Execution model
- Compilation pipeline
- Glue code generation
- Optimisation
- Results.

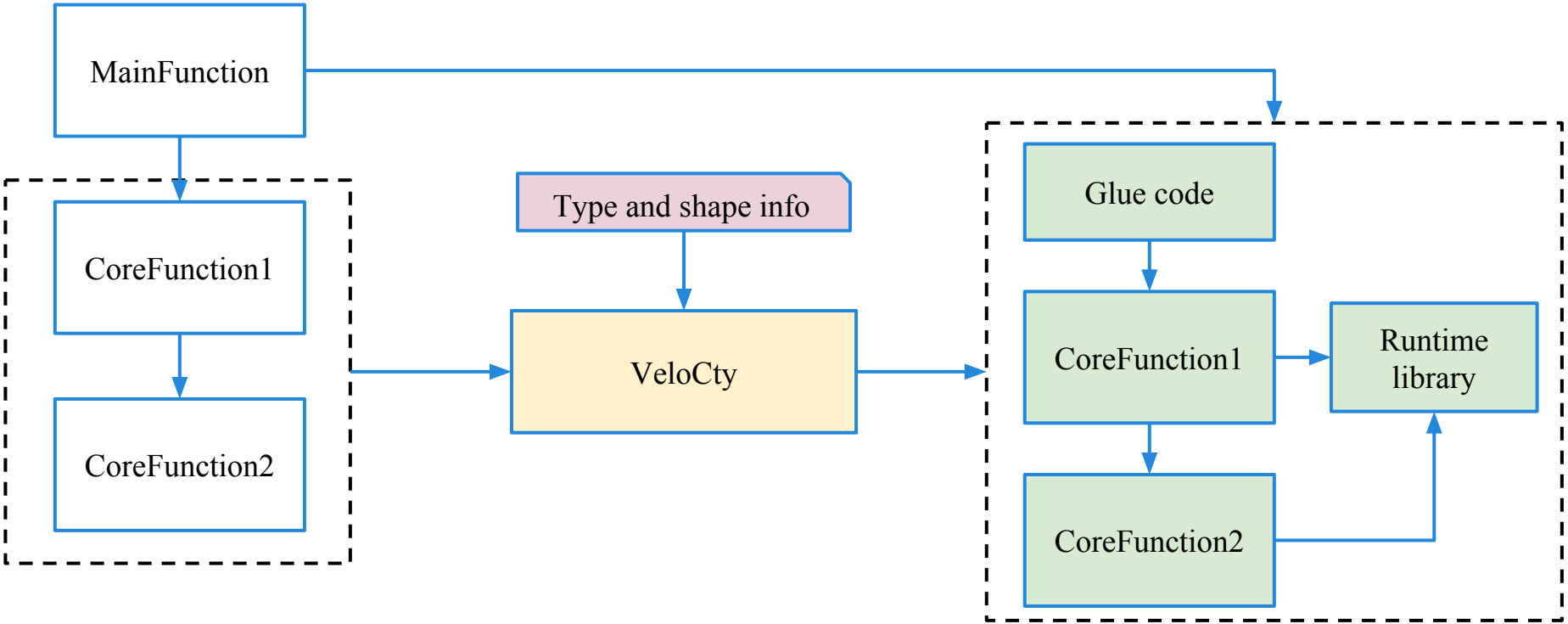
Background

- Velociraptor
  - Framework for generating high performance code for array-based languages.
- VRIR
  - High-level strongly typed AST.
  - Flexible array indexing.
  - Multiple array layouts.
- McLab
  - McLab Frontend
  - McSAF
  - Tamer and Tamer+

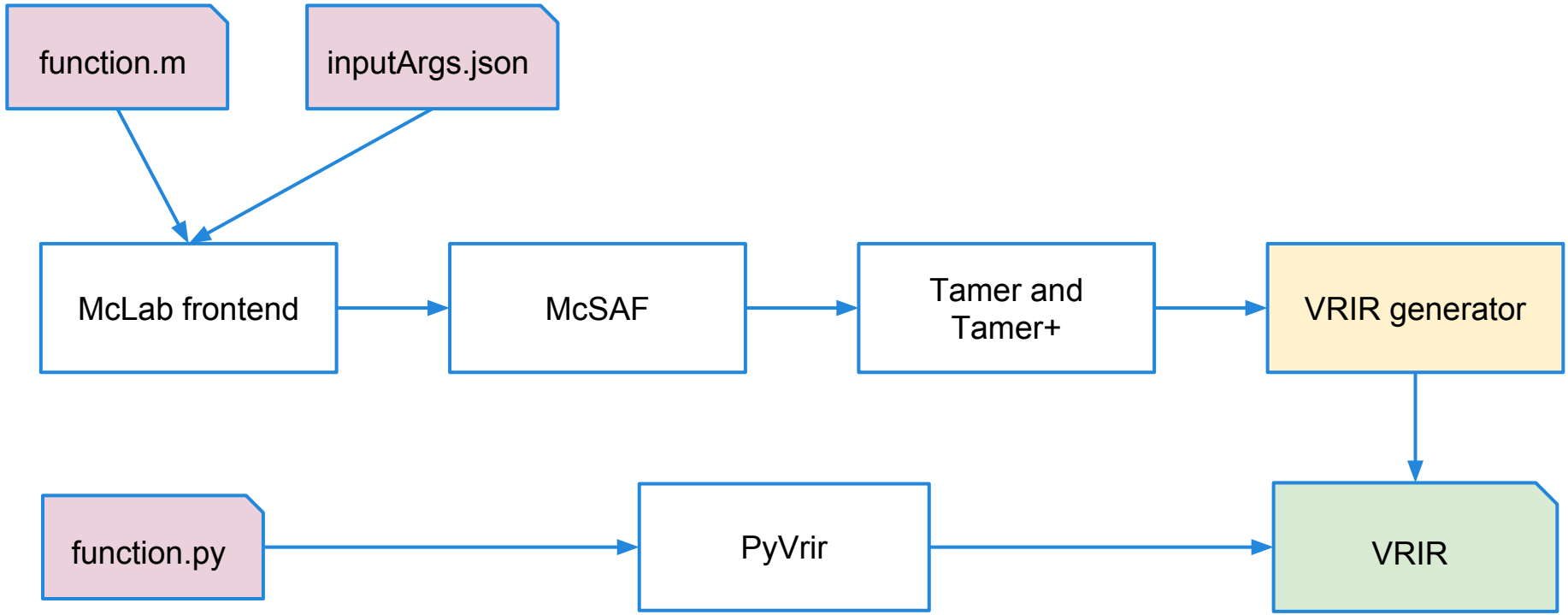
Introduction

- VeloCty is an optimising compiler for array-based languages to C++
- Matlab and Python's NumPy Library currently supported
- Supports parallelism using OpenMP.
- Provides a language specific runtime library.
- 1.1 to 400 times faster than Mathworks' Matlab.
- 47 to 541 times faster than C-Python interpreter

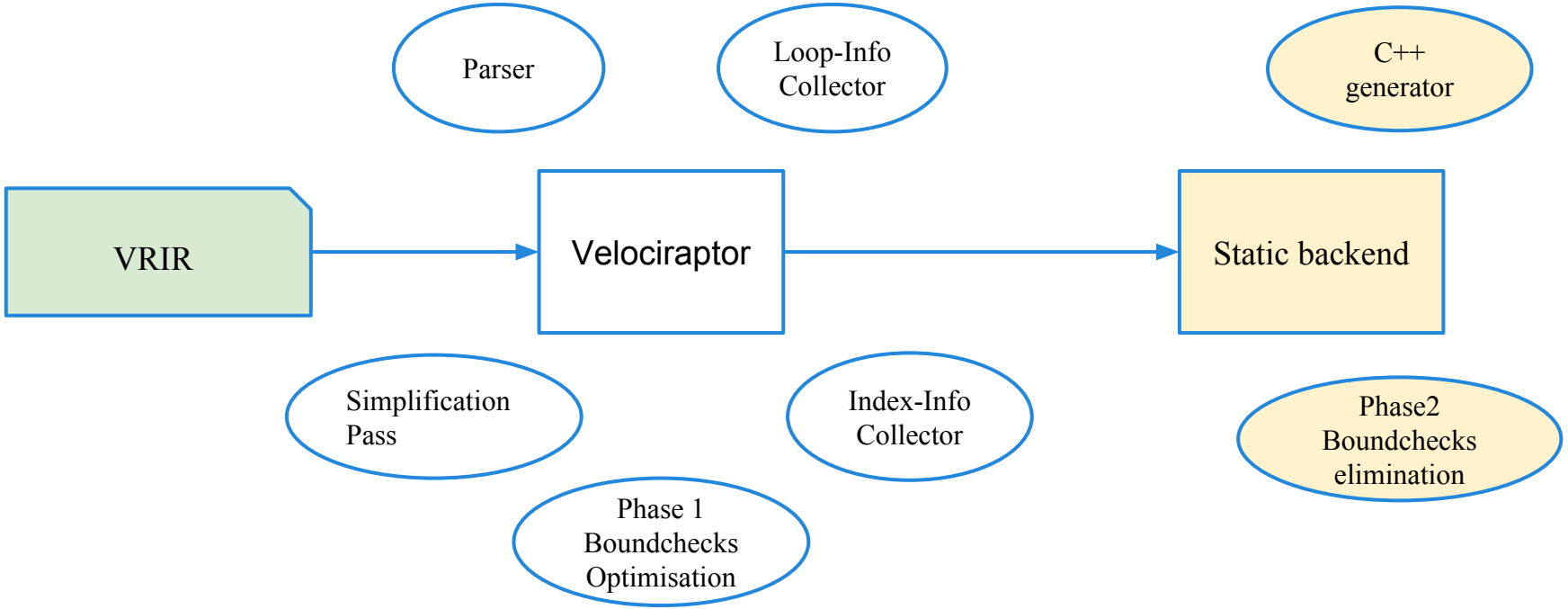
Execution Model



Compilation Pipeline



Compilation Pipeline



Example : Babai Nearest Plane Algorithm

```
function z_hat = babai(R,y)
        n=length(y);
       z_hat=zeros(n,1);
        z_hat(n)=round(y(n)./R(n,n));
       for k=n-1:-1:1
            par=R(k,k+1:n)*z_hat(k+1:n);
            ck=(y(k)-par)./R(k,k);
            z_hat(k)=round(ck);
       end
 end
```

Example : Babai Nearest Plane Algorithm

```
VrArrayPtrF64 babai(VrArrayPtrF64 R,VrArrayPtrF64 y){
    VrArrayPtrF64 z_hat;
    long n;
    long k;
    double par;
    double ck;
    double vr_temp7;
    double vr_temp8;
    double vr_temp9;
    VrArrayPtrF64 vr_temp10;
    VrArrayPtrF64 vr_temp11;
```

Array Type declarations

Scalar Type declarations

Example : Babai Nearest Plane Algorithm

```
n = length(y);
  z_hat = zeros_double(2,n,1);
 #ifdef BOUND_CHECK
    checkBounds_spec<VrArrayPtrF64,double>(&y,false,static_cast<dim_type>(n));
 #endif
   vr_temp7 = VR_GET_DATA_F64(y)[(n - 1)];
 #ifdef BOUND_CHECK
checkBounds_spec<VrArrayPtrF64,double>(&R,false,static_cast<dim_type>(n),static_cast<dim_type>(n));
 #endif
    vr_temp8 = VR_GET_DATA_F64(R)[(n - 1) + VR_GET_DIMS_F64(R)[0]*((n - 1))];
    vr_temp9 = (vr_temp7 / vr_temp8);
```

Example : Babai Nearest Plane Algorithm

```
#ifdef BOUND_CHECK
    checkBounds_spec<VrArrayPtrF64,double>(&z_hat,true,static_cast<dim_type>(n));
#endif

    VR_GET_DATA_F64(z_hat)[(n - 1)] = round(vr_temp9);
    for(k=(n - 1);k<= 1;k=k+(-1))
    {
      (R.sliceArraySpec(&vr_temp10,VrIndex(k),VrIndex((k + 1),n,1)));
       (z_hat.sliceArraySpec(&vr_temp11,VrIndex((k + 1),n,1)));
      BlasDouble::mmult(CblasColMajor,CblasNoTrans,CblasNoTrans,vr_temp10,vr_temp11, &par);
```

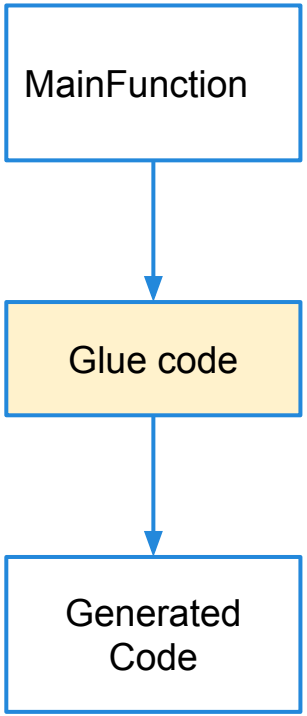
Example : Babai Nearest Plane Algorithm

```
#ifdef BOUND_CHECK
        checkBounds_spec<VrArrayPtrF64,double>(&y,false,static_cast<dim_type>(k));
#endif
        vr_temp12 = VR_GET_DATA_F64(y)[(k - 1)];
        vr_temp13 = (vr_temp12 - par);
#ifdef BOUND_CHECK
  checkBounds_spec<VrArrayPtrF64,double>(&R,false,static_cast<dim_type>(k),
static_cast<dim_type>(k));
#endif
       vr_temp14 = VR_GET_DATA_F64(R)[(k - 1) + VR_GET_DIMS_F64(R)[0]*((k - 1))];
        ck = (vr_temp13 / vr_temp14);
```

Example : Babai Nearest Plane Algorithm

```
#ifdef BOUND_CHECK
        checkBounds_spec<VrArrayPtrF64,double>(&z_hat,true,static_cast<dim_type>(k));
#endif
        VR_GET_DATA_F64(z_hat)[(k - 1)] = round(ck);
    }
    return z_hat;
}
```

Glue Code Generation



Glue Code Generation

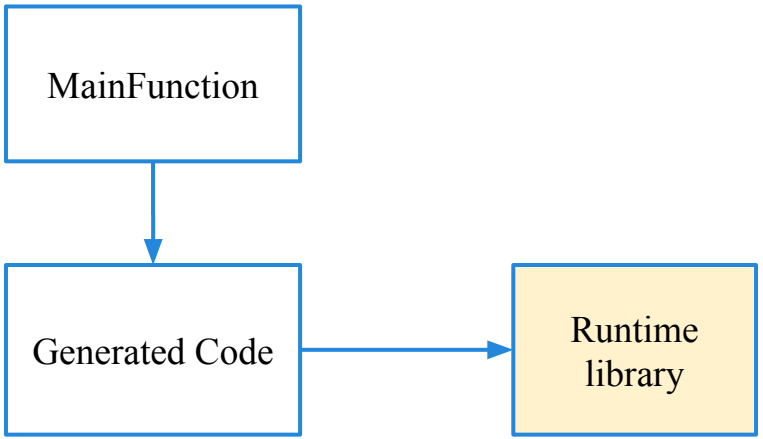
- Required for interfacing generated code with Matlab/Python.
- Mex API used for Matlab
- Python C/API.
- Glue generated automatically.

Glue Code Generation : Matlab

```
#include<mex.h>
#include"babaiImpl.hpp"

void mexFunction(int nlhs, mxArray *plhs[],
  int nrhs,const mxArray *prhs[]) {
  VrArrayF64 inputData0 = getVrArrayF64(rhs[0]);
  VrArrayF64 inputData1 = getVrArrayF64(rhs[1]);
  VrArrayF64 retVal = babai(inputData0,inputData1);
  plhs[0] = mxCreateNumericArray(retVal.ndims,retVal.dims, mxDOUBLE_CLASS, mxREAL);
  mxSetData(plhs[0],retVal.data);
}
```

Runtime Libraries



Runtime Libraries

- Implements structures to hold array data and metadata
- Supports various operations on arrays
  - Matrix multiply, addition etc.
  - Intel MKL used for Matlab and OpenBLAS for Python
- Supports builtins
  - Trigonometric functions, memory allocation functions etc.
  - Naive implementation.
- Implements arraybounds checks.
  - Supports array growth functions.

Optimisations

- Elimination of redundant memory allocations
- Bounds check eliminations.
- Parallel for loop generation.

Elimination of redundant memory allocations

- Memory allocated during array operations.
- Operations are inside loops.
- Optimisation replaces standard function with specialised function.
- Specialised function checks if sufficient memory is already allocated.
- Memory is reused if memory is sufficient.

Elimination of redundant memory allocations

```
drz = BlasDouble::scal_minus(Rz,vr_temp30);
```

```
BlasDouble::scal_minus(Rz,vr_temp30,&drz);
```

BoundChecks Elimination

- Phase 1 Boundchecks
  - Implemented inside Velociraptor
  - Eliminates redundant boundchecks.

- Phase 2 Boundchecks
  - Identifies affine indices inside for and parallel for loops
  - Moves checks outside the loop
  - Generates a check-free version of the loop
  - Check-free version is executed when all indices are within bounds.

Example : Phase 1 Checks

```
// LOOP BODY
#ifdef BOUND_CHECK
        checkBounds_spec<VrArrayPtrF64,double>(&c,false,static_cast<dim_type>(i),static_cast<dim_type>(j));
 #endif
        vr_temp9 = VR_GET_DATA_F64(c)[(i - 1) + VR_GET_DIMS_F64(c)[0]*((j - 1))];
 #ifdef BOUND_CHECK
         checkBounds_spec<VrArrayPtrF64,double>(&B,false,static_cast<dim_type>(h),static_cast<dim_type>(j));
 #endif
        vr_temp11 = VR_GET_DATA_F64(B)[(h - 1) + VR_GET_DIMS_F64(B)[0]*((j - 1))];
         vr_temp12 = (vr_temp10 * vr_temp11);
 #ifdef BOUND_CHECK
        checkBounds_spec<VrArrayPtrF64,double>(&c,true,static_cast<dim_type>(i),static_cast<dim_type>(j));
 #endif
         VR_GET_DATA_F64(c)[(i - 1) + VR_GET_DIMS_F64(c)[0]*((j - 1))] = (vr_temp9 + vr_temp12);
```

Example : Phase 2 Checks

```
if(checkDimStart_spec<VrArrayPtrF64>(c,1,1) && checkDimStop_spec<VrArrayPtrF64>(c,m,n) &&
checkDimStart_spec<VrArrayPtrF64>(B,1,1) && checkDimStop_spec<VrArrayPtrF64>(B,k,n) &&
checkDimStart_spec<VrArrayPtrF64>(A,1,1) && checkDimStop_spec<VrArrayPtrF64>(A,m,k)) {
        for(j=1;j<= n;j=j+static_cast<long>(1))
        {
           for(h=1;h<= k;h=h+static_cast<long>(1))
           {
              for(i=1;i<= m;i=i+static_cast<long>(1))
              {
                 vr_temp9 = VR_GET_DATA_F64(c)[(i - 1) + VR_GET_DIMS_F64(c)[0]*((j - 1))];
                 vr_temp10 = VR_GET_DATA_F64(A)[(i - 1) + VR_GET_DIMS_F64(A)[0]*((h - 1))];
                vr_temp11 = VR_GET_DATA_F64(B)[(h - 1) + VR_GET_DIMS_F64(B)[0]*((j - 1))];
                vr_temp12 = (vr_temp10 * vr_temp11);
                VR_GET_DATA_F64(c)[(i - 1) + VR_GET_DIMS_F64(c)[0]*((j - 1))] = (vr_temp9 +
vr_temp12);
              }
           }
        } }
```

Example: Phase 2 Checks

```
else {
        // Same Code but with checks added.
}
```

Parallel for loop generation

- Generated using OpenMP.
- User needs to provide list of shared variables.
- Generated for the parallel for statement inside VRIR.

Example Parallel for loops

```
#pragma omp parallel for shared(A,B,c) private(j,i,h,vr_temp9,vr_temp10,vr_temp11,vr_temp12)
 for(j=(1);j<= static_cast<long>(n);j=j+(1)) {


         for(h=1;h<= static_cast<long>(k);h=h+(1))
         {
            for(i= 1 ;i<= m; i = i + 1){
               <Loop Body>
             }
          }
 }
```

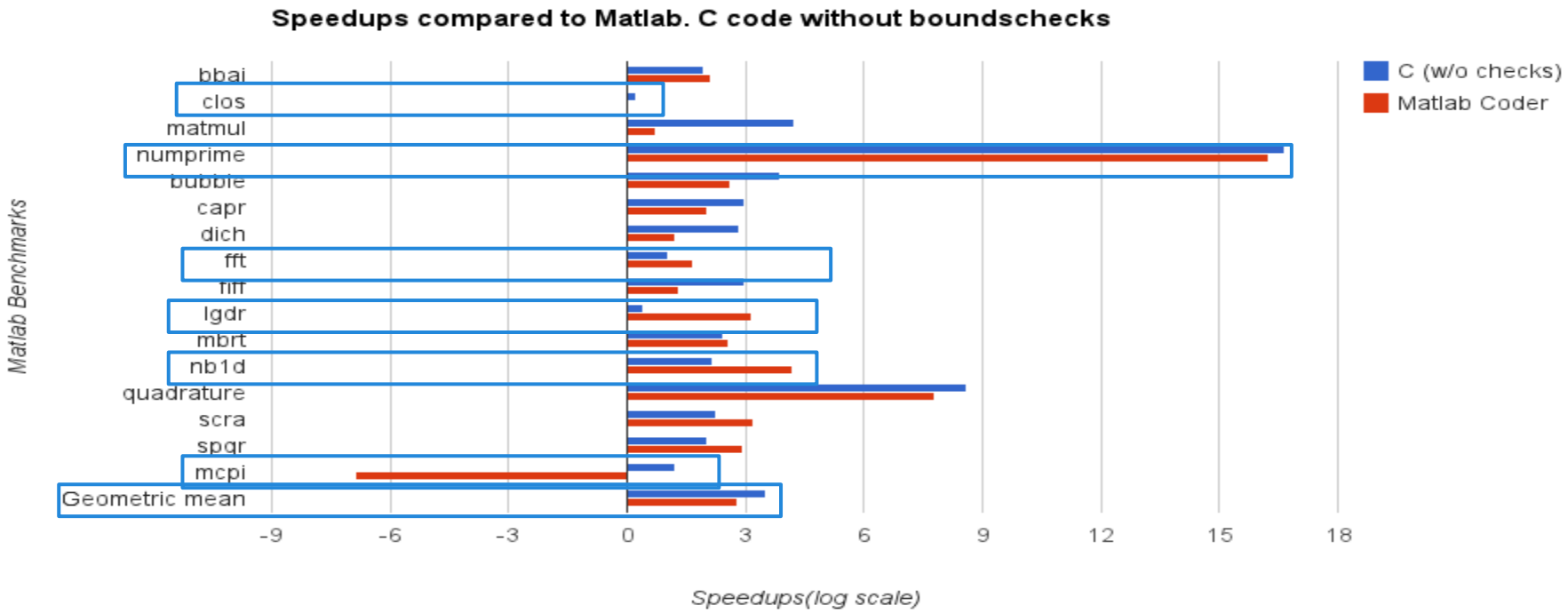
Experimental Results

- Matlab
  - Results obtained for 17 benchmarks
  - Compared against Mathworks' Matlab and Matlab coder
- Python
  - Results obtained for 9 Benchmarks.
  - Compared against Cython C-Python interpreter
- Different variations of the generated code compared.
  - Boundscheck optimisation turned on v/s turned off.
  - Memory allocation optimisation turned on v/s turned off.

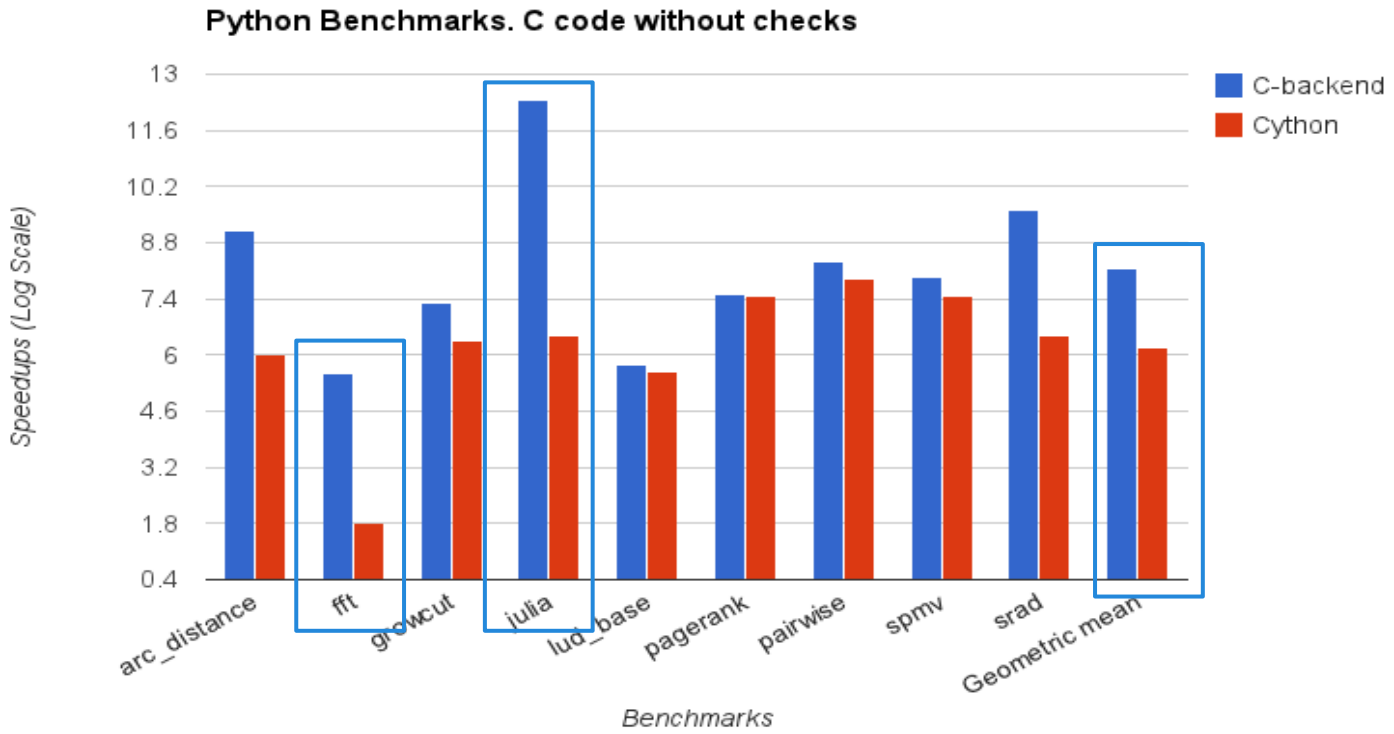
Machine Specifications

- Ubuntu 12.04
- 16 GB Physical memory
- Intel Core i7-3820 @3.60GHz

Matlab: C without boundschecks



Python : C without boundschecks



Matlab : C with checks v/s without checks

Slowdown of C with checks compared to C without checks	
Geometric mean (All Benchmarks)	Geometric mean (Affected benchmarks)
1.58	3.92

Python : C with checks v/s without checks

Slowdown of C with checks compared to C without checks	
Geometric mean (All Benchmarks)	Geometric mean (Affected benchmarks)
2.25	2.25

MATLAB : C boundscheck optimisation

Slowdown of C with optimisation compared to C without checks	
Geometric mean (All Benchmarks)	Geometric mean (Affected benchmarks)
1.03	1.02

MATLAB: C memory optimisation

Speedup of C with optimisation compared to C without checks	
Geometric mean (All Benchmarks)	Geometric mean (Affected benchmarks)
1.26	2.56

Future Work

- GPU code generation
- Faster builtin implementations
- Faster array slice operations

THANK YOU

https://github.com/Sable/VeloCty

Glue Code Generation : Python

```
static PyObject* julia_python_for_loops(PyObject* self, PyObject *args) {
      double obj1,obj2,obj4,obj5;
      int obj3,obj6;
      if(!PyArg_ParseTuple(args, "ddiddi", &obj1,&obj2, &obj3, &obj4,&obj5,&obj6)){
          return NULL;
      }
      VrArrayPtrI32 D = _julia_python_for_loops(obj1,obj2,obj3,obj4,obj5,obj6);
      long *dims = static_cast<long*>(malloc(sizeof(long)*D.ndims));
      for (int i = 0; i < D.ndims; i++) {
          dims[i] = D.dims[i];
      }
      PyArrayObject* ret = reinterpret_cast<PyArrayObject*>(PyArray_SimpleNewFromData(D.ndims, dims,NPY_INT32,D.
data));
      Py_INCREF(ret);
      return PyArray_Return(ret);
  }
```

Glue Code Generation : Python

```
static PyMethodDef juliaMethods[] =
  {
       {"julia_python_for_loops",julia_python_for_loops,METH_VARARGS, "Naive Matrix multiply"},
       {NULL,NULL,0,NULL}
   };
    static struct PyModuleDef juliaModule = {
       PyModuleDef_HEAD_INIT,
       "juliaModule",
       NULL,
       -1,
       juliaMethods,
   };
```