nb-l10-benchmark

December 8, 2019

Scientific Computing, TU Berlin, WS 2019/2020, Lecture 10

Jürgen Fuhrmann, WIAS Berlin

1 C vs Julia benchmark revisited

- During lecture 8, we had a discussion if the benchmark provided is fair
- Let us figure out…

1.1 Heat conduction problem from homework

$$\begin{aligned} -u'' &= 1 \quad \text{in } \Omega \\ -u'(0) + \alpha(u(0) - v_L) &= 0 \\ u'(1) + \alpha(u(1) - v_R) &= 0 \end{aligned}$$

- Assume $f = 1, v_L = 0, v_R = 0$

- Interior:

$$\begin{aligned} -u' &= x + C \\ u(x) &= -\frac{1}{2}x^2 - Cx + D \end{aligned}$$

- Left boundary condition:

$$\begin{aligned} -u'(0) + \alpha u(0) &= 0 \\ C + \alpha D &= 0 \\ C &= -\alpha D \end{aligned}$$

- Right boundary condition:

$$
\begin{aligned}
u'(1) + \alpha u(1) &= 0 \\
-1 - C + \alpha\left(-\frac{1}{2} - C + D\right) &= 0 \\
-1 + \alpha D + \alpha\left(-\frac{1}{2} + \alpha D + D\right) &= 0 \\
D(2\alpha + \alpha^2) &= \frac{1}{2}\alpha + 1 \\
\alpha D(2+\alpha) &= \frac{\alpha+2}{2} \\
D &= \frac{1}{2\alpha} \\
C &= -\frac{1}{2}
\end{aligned}
$$

- Solution:

$$u(x) = -\frac{1}{2}x^2 + \frac{1}{2}x + \frac{1}{2\alpha}$$

- Define solution array and exact solution

```
[1]: u(x,alpha)=0.5*(-x*x +x + 1/alpha)
     u_exact(N,alpha)=u.(collect(0:1/(N-1):1),alpha)
```

```
[1]: u_exact (generic function with 1 method)
```

1.2 Discrete problem from finite volume approximation

- gives a better idea how to handle boundary conditions...

$$
\begin{pmatrix}
\alpha + \frac{1}{h} & -\frac{1}{h} & & & & \\
-\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & & \\
 & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & \\
 & \ddots & \ddots & \ddots & \ddots & \\
 & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & \\
 & & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} \\
 & & & & -\frac{1}{h} & \frac{1}{h} + \alpha
\end{pmatrix}
\begin{pmatrix}
u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \\ u_N
\end{pmatrix}
=
\begin{pmatrix}
\frac{h}{2} \\ h \\ h \\ \vdots \\ h \\ h \\ \frac{h}{2}
\end{pmatrix}
$$

1.3 Problem setup

```
[2]: function setup(N,alpha)
         h=1.0/(N-1)
         a=[-1/h for i=1:N-1]
         b=[2/h for i=1:N]
         c=[-1/h for i=1:N-1]
```

```
    b[1]=alpha+1/h
    b[N]=alpha+1/h
    f=[h for i=1:N]
    f[1]=h/2
    f[N]=h/2
    return a,b,c,f
end
```

[2]:
```
setup (generic function with 1 method)
```

Correctness check

[3]:
```
check(N,alpha,solver)=norm(solver(setup(N,alpha)...)-u_exact(N,alpha))
```

[3]:
```
check (generic function with 1 method)
```

Setup tools

[4]:
```
using LinearAlgebra
using SparseArrays
using BenchmarkTools
```

1.4 Solvers

Progonka adapted from Daniel Kind, Alon Cohn

- “Clean” function without allocations
- We wil try some more optimizations suggested: @inbounds, @fastmath

[5]:
```
function progonka(u,a,b,c,f,Alpha,Beta)
    @inbounds @fastmath begin
        N = size(f,1)
    Alpha[2] = -c[1]/b[1]
    Beta[2] = f[1]/b[1]
    for i in 2:N-1 #Forward Sweep
        Alpha[i+1]=-c[i]/(a[i-1]*Alpha[i]+b[i])
        Beta[i+1]=(f[i]-a[i-1]*Beta[i])/(a[i-1]*Alpha[i]+b[i])
    end
    u[N]=(f[N]-a[N-1]*Beta[N])/(a[N-1]*Alpha[N]+b[N])
    for i in N-1:-1:1 #Backward Sweep
        u[i]=Alpha[i+1]*u[i+1]+Beta[i+1]
    end
    end
end
```

[5]:
```
progonka (generic function with 1 method)
```

Wrapper with allocations

[6]:

```
function julia_progonka(a,b,c,f)
    N = size(f,1)
    u=Vector{eltype(a)}(undef,N)
    Alpha=Vector{eltype(a)}(undef,N)
    Beta=Vector{eltype(a)}(undef,N)
    progonka(u,a,b,c,f,Alpha,Beta)
    return u
end
```

[6]:

```
julia_progonka (generic function with 1 method)
```

Setup data

[7]:

```
alpha=1
N=1000
a,b,c,f=setup(N,alpha)
```

[7]:

```
([-999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0
… -999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0, -999.0,
-999.0], [1000.0, 1998.0, 1998.0, 1998.0, 1998.0, 1998.0, 1998.0, 1998.0,
1998.0, 1998.0  …  1998.0, 1998.0, 1998.0, 1998.0, 1998.0, 1998.0, 1998.0,
1998.0, 1998.0, 1000.0], [-999.0, -999.0, -999.0, -999.0, -999.0, -999.0,
-999.0, -999.0, -999.0, -999.0  …  -999.0, -999.0, -999.0, -999.0, -999.0,
-999.0, -999.0, -999.0, -999.0, -999.0], [0.0005005005005005005,
0.001001001001001001, 0.001001001001001001, 0.001001001001001001,
0.001001001001001001, 0.001001001001001001, 0.001001001001001001,
0.001001001001001001, 0.001001001001001001, 0.001001001001001001  …
0.001001001001001001, 0.001001001001001001, 0.001001001001001001,
0.001001001001001001, 0.001001001001001001, 0.001001001001001001,
0.001001001001001001, 0.001001001001001001, 0.001001001001001001,
0.0005005005005005005])
```

Check correctness of solution

[8]:

```
@show check(N,alpha,julia_progonka)
```

```
check(N, alpha, julia_progonka) = 4.131805909999797e-12
```

[8]:

```
4.131805909999797e-12
```

Benchmark

[9]:

```
@btime julia_progonka(a,b,c,f);
```

```
  6.575  s (3 allocations: 23.81 KiB)
```

Progonka in C

- Create file progonka.c

[10]:

```
open("progonka.c", "w") do io
    write(io, """
#include <time.h>

void progonka(int N,double* u,double* a,double* b,double* c,double* f,double* Alpha,double* Beta)
{
  int i;
  /* Adjust indexing:
    This is C pointer arithmetic. Shifting the start addresses by 1
    allows to keep the indexing from 1.
 */
  u--;
  a--;
  b--;
  c--;
  f--;
  Alpha--;
  Beta--;
  Alpha[2] = -c[1]/b[1];
  Beta[2] = f[1]/b[1];
  for(i=2;i<=N-1;i++)
  {
    Alpha[i+1]=-c[i]/(a[i-1]*Alpha[i]+b[i]);
    Beta[i+1]=(f[i]-a[i-1]*Beta[i])/(a[i-1]*Alpha[i]+b[i]);
  }
  u[N]=(f[N]-a[N-1]*Beta[N])/(a[N-1]*Alpha[N]+b[N]);
  for(i=N-1;i>=1;i--)
  {
    u[i]=Alpha[i+1]*u[i+1]+Beta[i+1];
  }
}

double tmem; /* time memory variable */

void tstart(void) /* Start time measurement */
{
  tmem=(double)clock()/(double)CLOCKS_PER_SEC;
}

void tstop(void) /* Stop time measurement */
{
  tmem=(double)clock()/(double)CLOCKS_PER_SEC-tmem;
}

double tget(void) /* Return value of timer */
{
  return tmem;
}

/* Measure time in C. Call one million times. */
void c_progonka_with_timing(int N,double* u,double* a,double* b,double*␣
 ↪c,double* f,double* Alpha,double* Beta)
{
  int itime;
  int ntime;
  ntime=1000000;
  tstart();
  for(itime=0;itime<ntime;itime++)
  {
    progonka(N,u,a,b,c,f,Alpha,Beta);
  }
  tstop();
}
""")
end
```

[10]:
```
1245
```

- Compile file progonka.c with highest optimization level
- Suggested further optimizations: -march=native
- Possibly try different compiler

[11]:
```
run(`clang -fPIC -Ofast -march=native --shared progonka.c -o progonka.so`)
```

[11]:
```
Process(`clang -fPIC -Ofast -march=native
--shared progonka.c -o progonka.so`,
ProcessExited(0))
```

- Wrap C timer calls for use from Julia

[12]:
```
tstart()=ccall( (:tstart,"progonka"),Cvoid,())
tstop()=ccall( (:tstop,"progonka"),Cvoid,())
tget()=ccall( (:tget,"progonka"),Cdouble,())
```

[12]:
```
tget (generic function with 1 method)
```

- Julia wrapper for C code

[13]:
```
function c_progonka(a,b,c,f)
    u=Vector{eltype(a)}(undef,N)
    Alpha=Vector{eltype(a)}(undef,N)
    Beta=Vector{eltype(a)}(undef,N)
    ccall( (:progonka,"progonka"),␣
 ↪Cvoid,(Cint,Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},P
            N,u,a,b,c,f,Alpha,Beta)
    return u
end
```

[13]:
```
c_progonka (generic function with 1 method)
```

[14]:
```
@show check(N,alpha,c_progonka)
@btime c_progonka(a,b,c,f);
```

```
check(N, alpha, c_progonka) = 4.131805909999797e-12
  8.895  s (8 allocations: 23.89 KiB)
```

- Driver for Julia progonka with timing

[15]:
```
function julia_progonka_with_timing(a,b,c,f)
    N = size(f,1)
    u=Vector{eltype(a)}(undef,N)
    Alpha=Vector{eltype(a)}(undef,N)
    Beta=Vector{eltype(a)}(undef,N)
    tstart()
    for itime=1:1000000
        progonka(u,a,b,c,f,Alpha,Beta)
    end
    tstop()
    return u
end
```

[15]:
```
julia_progonka_with_timing (generic function with 1 method)
```

[16]:
```
julia_progonka_with_timing(a,b,c,f)
print("time per call: $(tget()) s")
```

```
time per call: 6.239876999999998 s
```

- Julia wrapper for C code with C based timer

[17]:
```
function c_progonka_with_timing(a,b,c,f)
    u=Vector{eltype(a)}(undef,N)
    Alpha=Vector{eltype(a)}(undef,N)
    Beta=Vector{eltype(a)}(undef,N)
    ccall( (:c_progonka_with_timing,"progonka"),␣
 ↪Cvoid,(Cint,Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},P
            N,u,a,b,c,f,Alpha,Beta)
end
```

[17]:
```
c_progonka_with_timing (generic function with 1 method)
```

[18]:
```
c_progonka_with_timing(a,b,c,f)
print("time per call: $(tget())  s")
```

```
time per call: 5.353897999999999  s
```

- Julia wrapper for C code timed from Julia including ccall overhead

[19]:
```
function c_progonka_with_timing_from_julia(a,b,c,f)
    u=Vector{eltype(a)}(undef,N)
    Alpha=Vector{eltype(a)}(undef,N)
    Beta=Vector{eltype(a)}(undef,N)
    tstart()
    for itime=1:1000000
        ccall( (:progonka,:progonka),␣
 ↪Cvoid,(Cint,Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},Ptr{Cdouble},P
                      N,u,a,b,c,f,Alpha,Beta)
    end
    tstop()
end
```

[19]:
```
c_progonka_with_timing_from_julia (generic function with 1 method)
```

[20]:
```
c_progonka_with_timing_from_julia(a,b,c,f)
print("time per call: $(tget())  s")
```

```
time per call: 5.696459999999999  s
```

This notebook was generated using Literate.jl.