A Semi-Implicit numerical method for Discontinuous Hermite Collocation on GPUs

Nikolaos D. Vilanakis

Co-Author : *Emmanuel N. Mathioudakis*

TECHNICAL UNIVERSITY OF CRETE
APPLIED MATHEMATICS AND COMPUTERS LABORATORY
73100 CHANIA - CRETE - GREECE

ΘΑΛΗΣ

European Union
European Social Fund

OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS
MANAGING AUTHORITY

NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND

Co-financed by Greece and the European Union

Talk Overview

- Development of a parallel algorithm for the Discontinuous Hermite Collocation method on Parallel Architectures with accelerators
- Parallel implementation on computing architectures with GPUs

Semi-Implicit Discontinuous Hermite Collocation Method

INPUT $\boldsymbol{a}_{old}$, B , A_0 , A , A_b

for $t = dt$ *to* t_{max} *with step* dt *do*

    compute $\boldsymbol{a}_0 = A_0\, \boldsymbol{a}_{old}$

    if $t \le 2\, dt$ *then*

        solve $A_b\, \boldsymbol{a}_{new} = \boldsymbol{a}_{old}$ *with BiCGSTAB*

    else

        solve $A\, \boldsymbol{a}_1 = \boldsymbol{a}_0$ *with BiCGSTAB*

        compute $\boldsymbol{a}_0 = -dt\frac{\sqrt{3}}{3} B\, \boldsymbol{a}_1$

        solve $A\, \boldsymbol{a}_2 = \boldsymbol{a}_0$ *with BiCGSTAB*

        compute $\boldsymbol{a}_2 = \frac{dt}{2} B\, (\boldsymbol{a}_1 + \boldsymbol{a}_2)$

        solve $A_0\, \boldsymbol{a}_{new} = \boldsymbol{a}_2$ *with BiCGSTAB*

    endif

    compute $\boldsymbol{a}_{old} = \boldsymbol{a}_{new}$

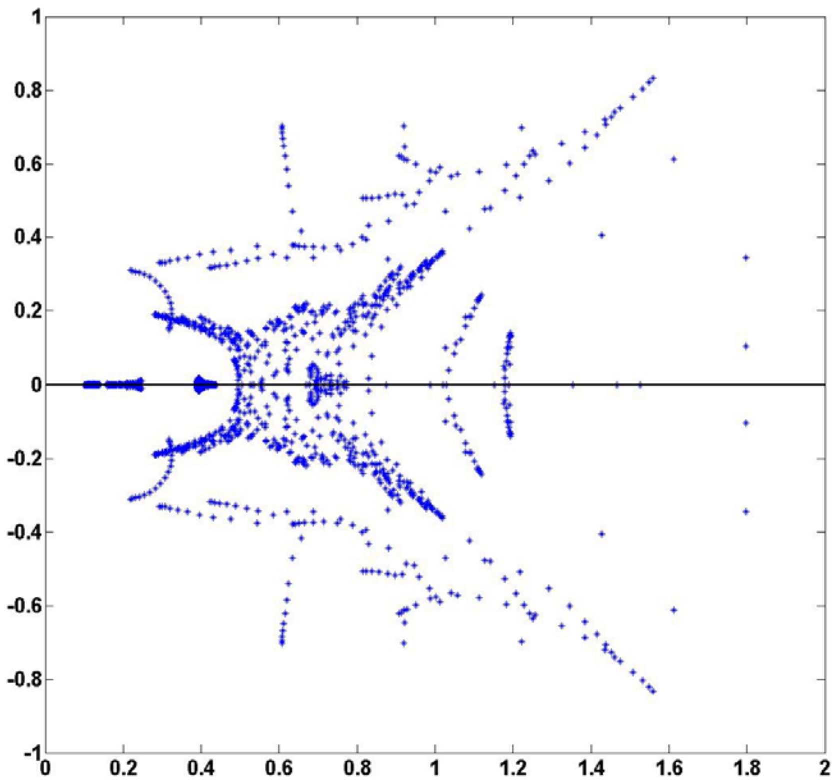
enddo

Preconditioned BiCGSTAB Method…

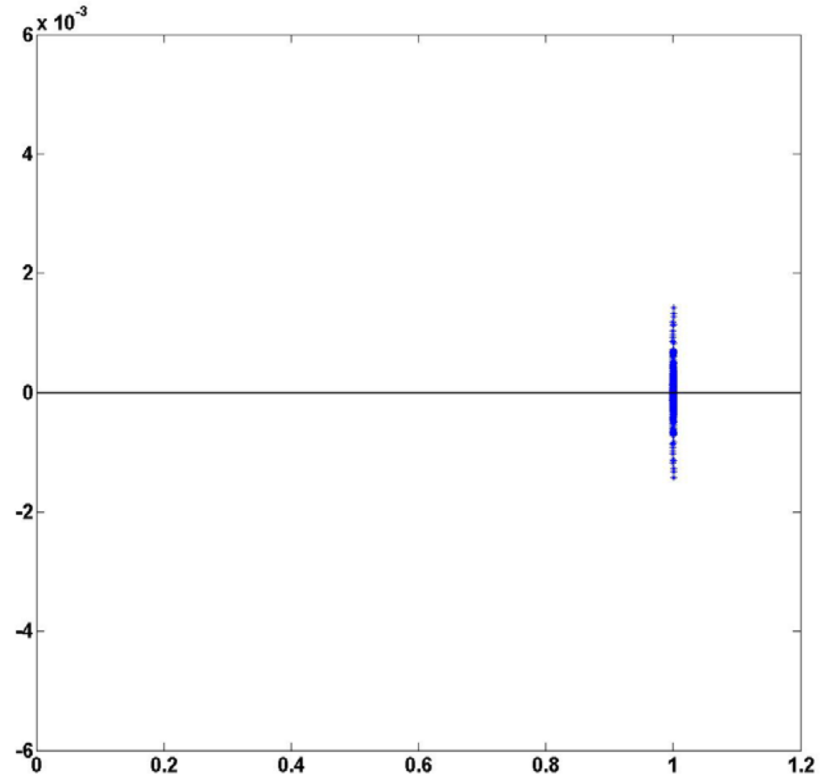
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
Choose $\tilde{r}$ (for example, $\tilde{r} = r^{(0)}$)
for $i = 1, 2, \ldots$
    $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
    if $\rho_{i-1} = 0$ **method fails**
    if $i = 1$
        $p^{(i)} = r^{(i-1)}$
    else
        $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
        $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1}v^{(i-1)})$
    endif
    solve $M\hat{p} = p^{(i)}$
    $v^{(i)} = A\hat{p}$
    $\alpha_i = \rho_{i-1}/\tilde{r}^T v^{(i)}$
    $s = r^{(i-1)} - \alpha_i v^{(i)}$
    check norm of s; if small enough: set $x^{(i)} = x^{(i-1)} + \alpha_i\hat{p}$ and stop
    solve $M\hat{s} = s$
    $t = A\hat{s}$
    $\omega_i = t^T s / t^T t$
    $x^{(i)} = x^{(i-1)} + \alpha_i\hat{p} + \omega_i\hat{s}$
    $r^{(i)} = s - \omega_i t$
    check convergence; continue if necessary
    for continuation it is necessary that $\omega_i \neq 0$
end

$$A \sim M = ilu(A)$$

Matrix A_b

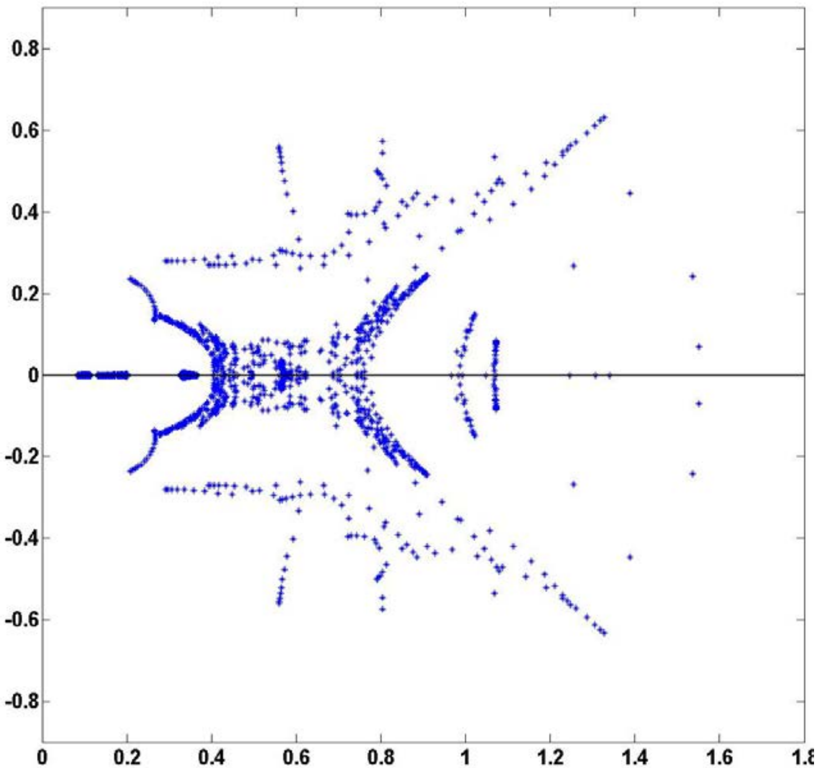


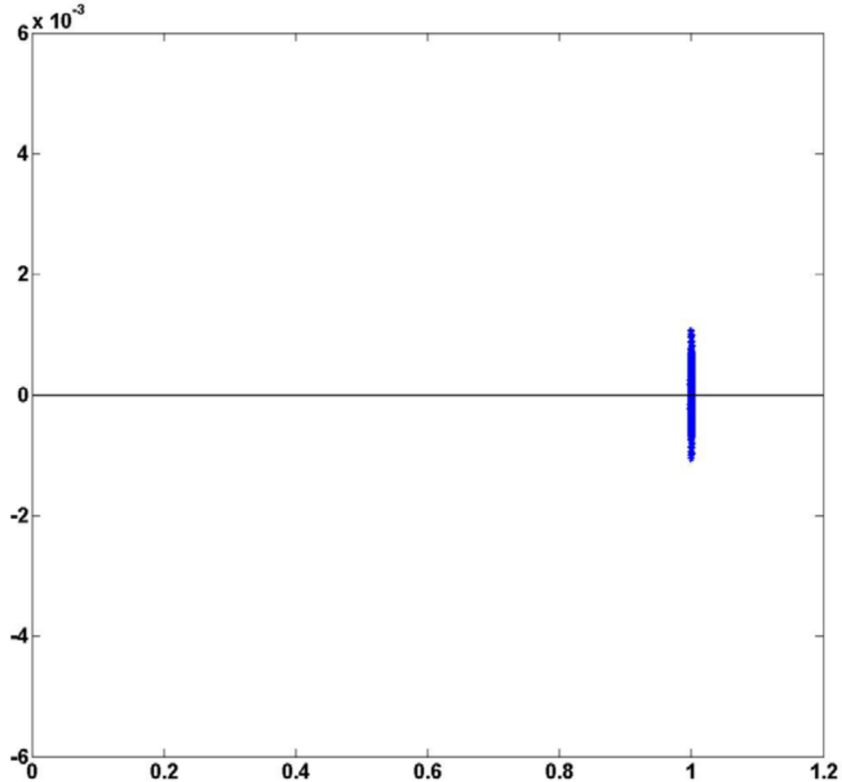
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2



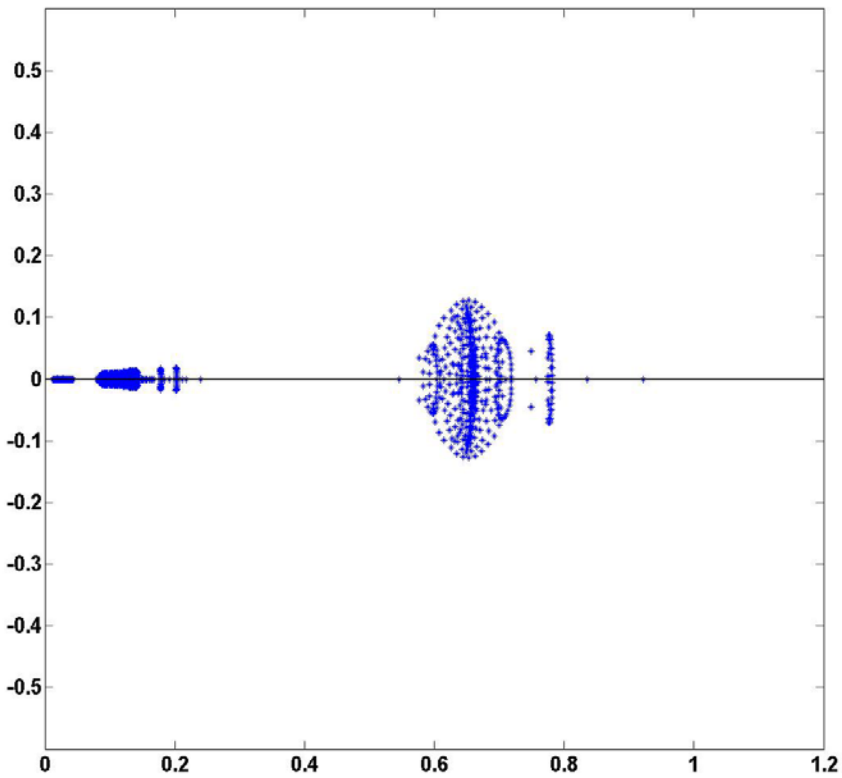
x 10^-3
6
4
2
0
-2
-4
-6
0 0.2 0.4 0.6 0.8 1 1.2

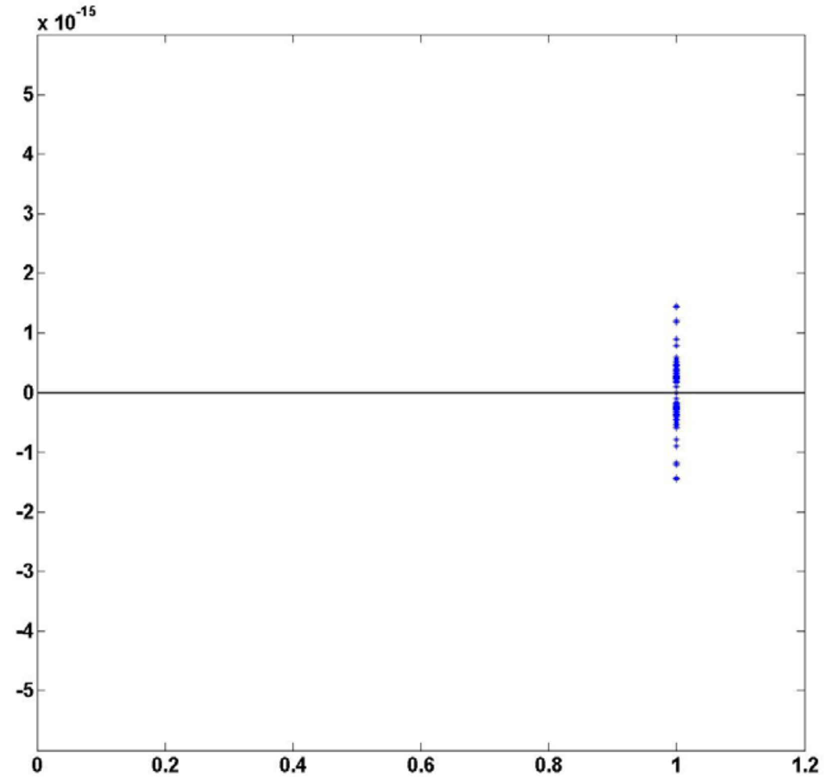
Matrix A


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8


x 10^-3
6
4
2
0
-2
-4
-6
0
0.2
0.4
0.6
0.8
1
1.2

Matrix A_0


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
0.2
0.4
0.6
0.8
1
1.2


x 10-15
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
0.2
0.4
0.6
0.8
1
1.2

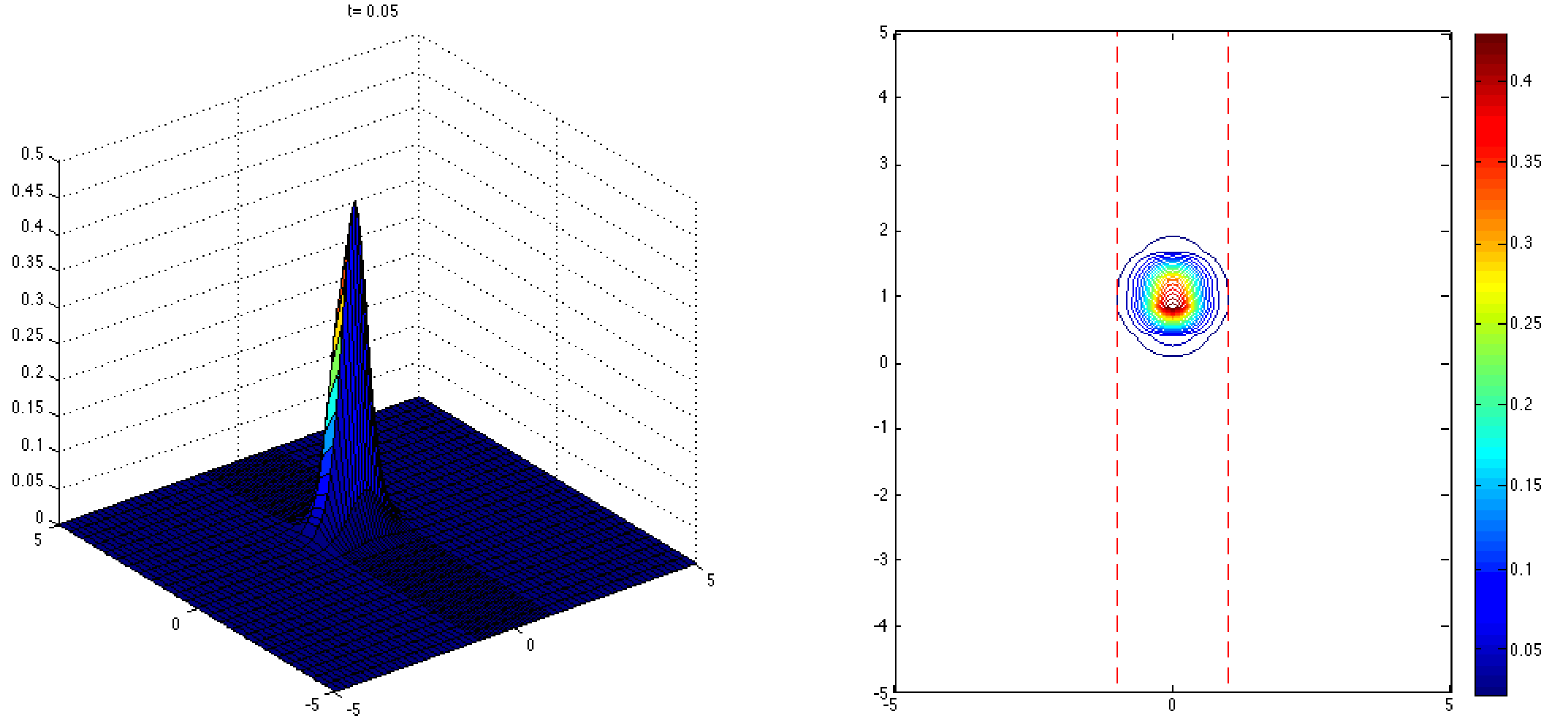
Parallel BiCGSTAB

- ✓ All basic linear algebra operations are performed on the GPU

- ✓ The preconditioning procedure $Mz = t$ with $M=LU$ is performed on the CPU

  Step 1 : GPU sends to CPU vector t
  Step 2 : CPU solution of $Ly = t$
  Step 3 : CPU solution of $Uz = y$
  Step 4 : CPU sends to GPU vector z

The test problem …



HP SL390s - Tesla M2070 GPUs

HP SL390s

+

6 core Xeon@2.8GHz
24GB memory
Oracle Linux 6.3 x64
PGI 14.5 Cuda Fortran
Cuda toolkit 6.0
PCI-e gen2 x16

The Portland Group

PGI Accelerator Compilers

TECHNICAL SPECIFICATIONS

	Tesla M2070 / M2075
Peak double precision floating point performance	515 Gigaflops
Peak single precision floating point performance	1030 Gigaflops
CUDA cores	448
Memory size (GDDR5)	6 GigaBytes
Memory bandwidth (ECC off)	150 GBytes/sec

NVIDIA® CUDA™
Parallel Programming and Computing Platform
NVIDIA.

Development tools …

- MatLab R2012b
- PGI 14.5 Cuda Fortran
  Cuda toolkit 6.0 – cuBLAS - cuSRARSE
  for GPU operations
  SparseKit for CPU operations

```
   module bicgstabGPU
   contains
    subroutine bicgstabGPU(n,x,b,A,L,U,iA,iL,jA,jL,jU,
+                            nzA,descrA,handle,error,r,rh,pi,ph,
+                            t,s,sh,ui,istep,temp,ttt)
 implicit real*8 (a-h,o-z)
 real*8 ttt(n),L(*),U(*),temp(n)
 integer*8 handle,descrA,iA,jA,A,
+             x,b,r,rh,pi,ph,s,sh,ui,t
 integer   cusparse_dcsrmv,iL(n+1),jL(*),jU(*),
+   cuda_memcpy_c2fort_real,cuda_memcpy_fort2c_real


 imaxstep=istep
 tol=error
 istep=0
 dnrmb=cublas_dnrm2(n,b,1)
 call cublas_dcopy(n,b,1,x,1)
 istatus=cusparse_dcsrmv(handle,0,n,n,nzA,1.0d0,descrA,A,iA,jA,
+          x,0.0d0,r)
```

```
      call cublas_dscal(n,-1.0d0,r,1)
      call cublas_daxpy(n,1.0d0,b,1,r,1)
      call cublas_dcopy(n,r,1,rh,1)
999   continue
      istep=istep+1
      if (istep.gt.1) roip2=roip1
       roip1=cublas_ddot(n,rh,1,r,1)
      if (istep.eq.1) then
       call cublas_dcopy(n,r,1,pi,1)
      else
       bi=(roip1/roip2)*(ai/wi)
       call cublas_daxpy(n,-wi,ui,1,pi,1)
       call cublas_dcopy(n,r,1,t,1)
       call cublas_daxpy(n,bi,pi,1,t,1)
       call cublas_dcopy(n,t,1,pi,1)
      endif
c-----------------------------------------
```

```
      icudaStat = cuda_memcpy_c2fort_real(ttt,pi,n*8,2)
      call lsol(n,temp,ttt,L,jL,iL)
      call udsol(n,ttt,temp,U,jU)
      icudaStat3 = cuda_memcpy_fort2c_real(ph,ttt,n*8,1)
c-----------------------------------------
      istatus=cusparse_dcsrmv(handle,0,n,n,nzA,1.0d0,descrA,A,iA,jA,
     +         ph,0.0d0,ui)
      ai=roip1/cublas_ddot(n,rh,1,ui,1)
      call cublas_dcopy(n,r,1,s,1)
      call cublas_daxpy(n,-ai,ui,1,s,1)
      if (cublas_dnrm2(n,s,1).lt.1.0d-25) then
      call cublas_daxpy(n,ai,ph,1,x,1)
       print*,'BiCGSTAB incomplete'
      else
c----------------------------------------
       icudaStat = cuda_memcpy_c2fort_real(ttt,s,n*8,2)
       call lsol(n,temp,ttt,L,jL,iL)
       call udsol(n,ttt,temp,U,jU)
       icudaStat3 = cuda_memcpy_fort2c_real(sh,ttt,n*8,1)
```

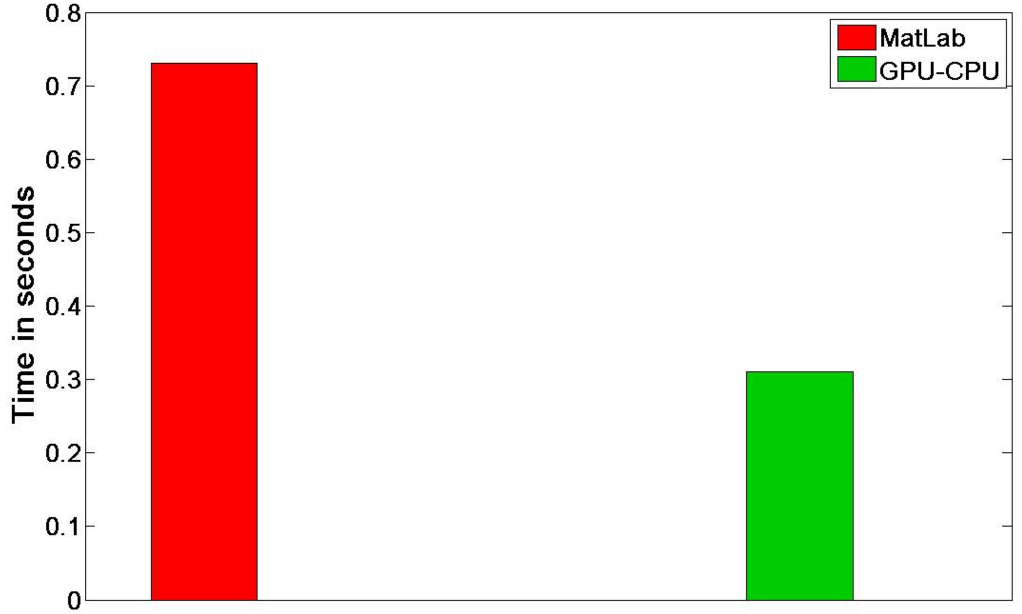
```
c-----------------------------------------
      istatus=cusparse_dcsrmv(handle,0,n,n,nzA,1.0d0,descrA,A,iA,jA,
     +         sh,0.0d0,t)
      wi=cublas_ddot(n,t,1,s,1)/cublas_ddot(n,t,1,t,1)
      call cublas_daxpy(n,ai,ph,1,x,1)
      call cublas_daxpy(n,wi,sh,1,x,1)
      call cublas_daxpy(n,-wi,t,1,s,1)
      call cublas_dcopy(n,s,1,r,1)
       dnrmr=cublas_dnrm2(n,s,1)
       error=dnrmr/dnrmb
c      print*,error,istep
      if (wi.ne.0.0d0.and.error.gt.tol.and.istep.lt.imaxstep)
     + goto 999
      endif
      return
      end
      end  module
```

Realization on HP SL390s Tesla GPU machine
Time measurements

Finite Elements : 400

Unknowns : 1.600

Dofs : 6.400

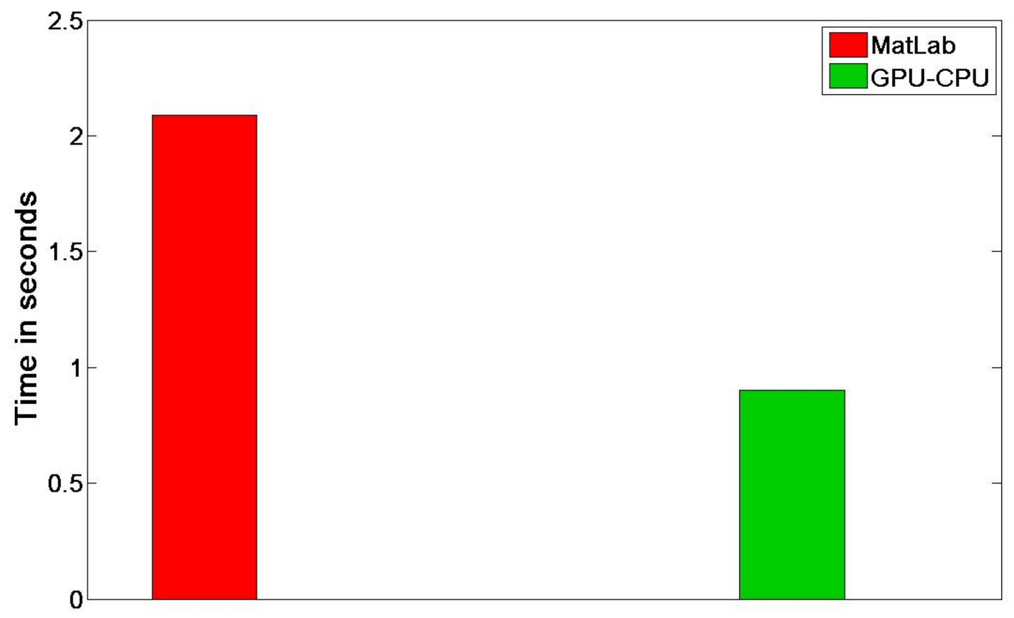


Realization on HP SL390s Tesla GPU machine
Time measurements

Finite Elements : 1.600

Unknowns : 6.400

Dofs : 25.600

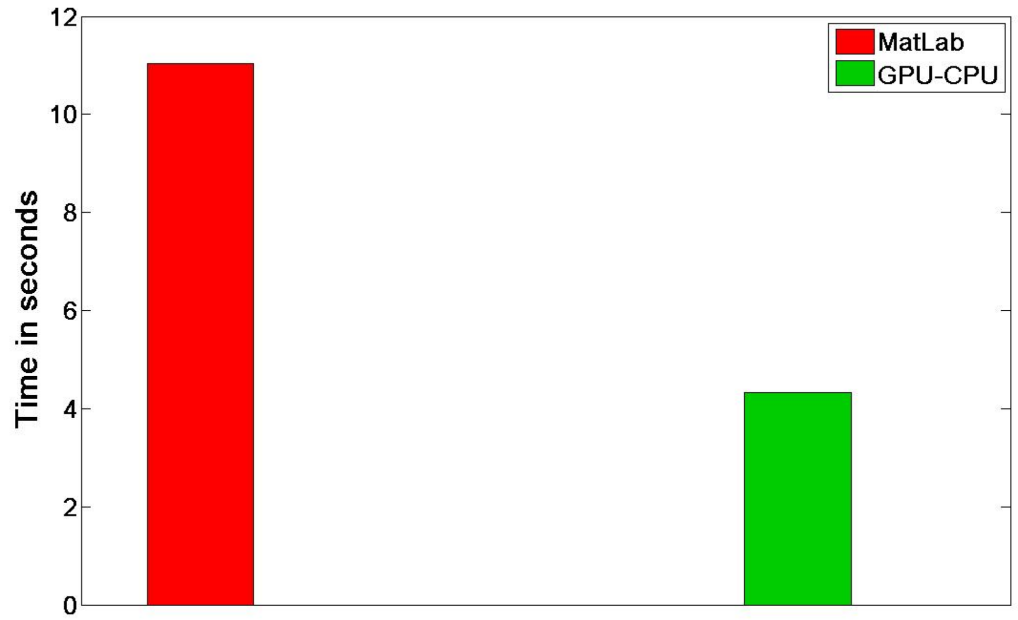


Realization on HP SL390s Tesla GPU machine
Time measurements

Finite Elements : 6.400

Unknowns : 25.600

Dofs : 102.400

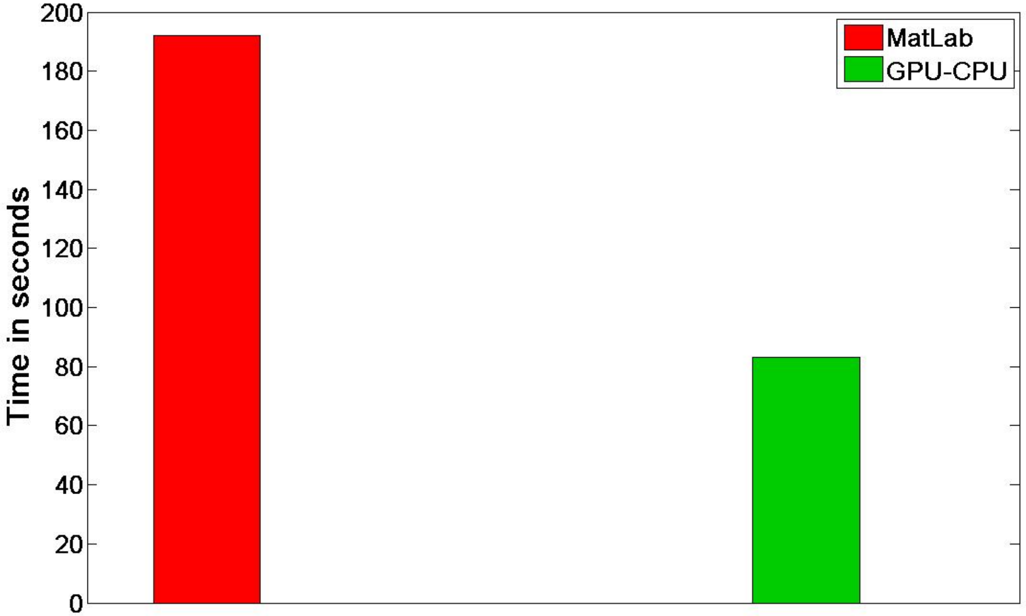


Realization on HP SL390s Tesla GPU machine
Time measurements

Finite Elements : 25.600

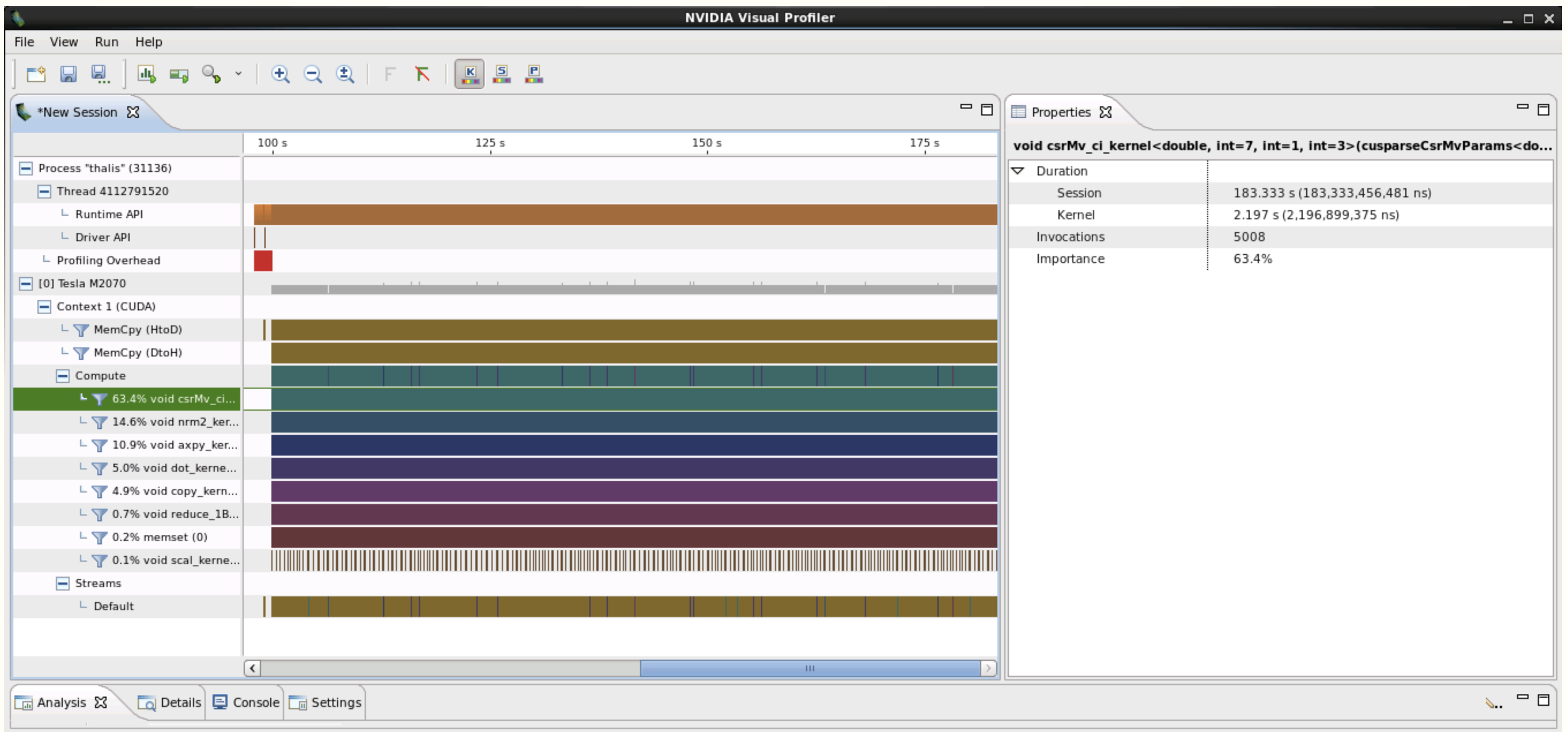
Unknowns : 102.400

Dofs : 409.600



Realization on HP SL390s Tesla GPU machine
Time measurements

Finite Elements : 25.600
Unknowns : 102.400
Dofs : 409.600

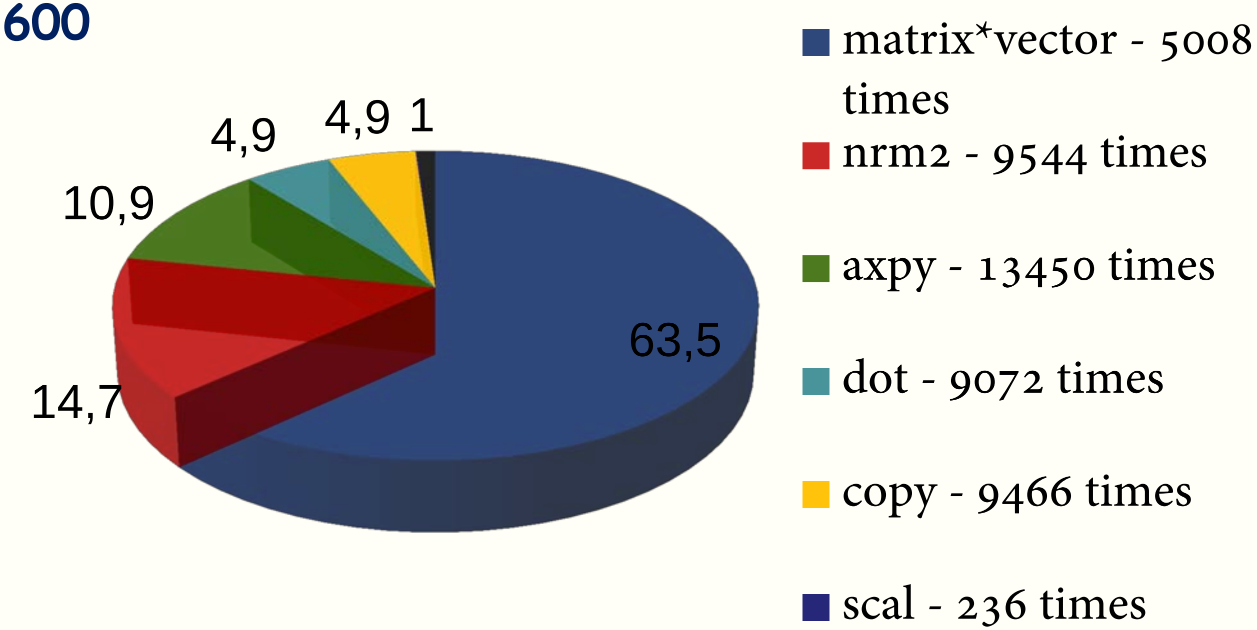


Realization on HP SL390s Tesla GPU machine
Time measurements

Finite Elements : 25.600

Unknowns : 102.400

Dofs : 409.600



Conclusions

- A new parallel algorithm implementing the Discontinuous Hermite Collocation method has been developed.
- The algorithm is realized on machines with GPU accelerators .
- A performance acceleration of up to 50% is observed for fine discretizations over MatLab multithread implementations.

Future work

- Design an efficient parallel algorithm for machines with multiple GPUs using cuBLAS-XT.