1 Gliederung

- ABSNF
- Problemstellung
- Project Structure (where is what)
- Devices
- Eval
- Gradient
- Solve
- Grid and Blocksize
- improvements
  - solve algorithmus
  - sparsity
  - multidevice support
  - memorymangaer

1.0.1 Kapitel

- Problemstellung
- Implementierung
- Performance
- Improvements and bottlenecks

2 Introduction

- Einführung ABSNF
- Aufgabenstellung
- Wichtige Fragen
- Was habe ich gemacht
- Used Libraries
- Devices
- Notation and Symbols
- Description of the data structures
- elementwise abs

3 Evalutation of the ABSNF

- Problem
- Implementations
- Performance
  - Single Core Float and Double
  - GTX Float and Double
  - Tesla Float and Double
- Review and Notes

4 Gradient of the ABSNF

5 Grid and Blocksize

6 Solve

7 Final Thoughts

- Improvements
- view
- Multidevice support
- Grid and Blocksize

8 Anhang

- projectstructure (unittests ect, python prototypes)
- cublas check

9 Evaluation of the ABSNF

9.1 Problem Specification

ABS-Normal Form:

$$\begin{pmatrix} \Delta z \\ \Delta y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} Z & L \\ J & Y \end{pmatrix} \times \begin{pmatrix} \Delta x \\ |\Delta z| \end{pmatrix}$$

Given is a PL function in abs-nf. The evaluation of this function means calculating the vectors Δy and Δz:

$$\Delta y = b + (J \times \Delta x) + (Y \times |\Delta z|) \tag{1}$$
$$\Delta z = a + (Z \times \Delta x) + (L \times |\Delta z|) \tag{2}$$

where the following structures are given:

$$a, b, Z, L, J, Y, m, n, s, \Delta x$$

In (2) Δz depends on the element-wise absolute function of its own and therefore it cannot be calculated straightforward. Since the matrix L is lower triangular, the vector Δz can be iteratively calculated, by taking the row-wise dotproduct of L and $|\Delta z|$.

$$k = a + Z \times \Delta x$$

$$\begin{aligned}
\boxed{\Delta z_1} &= \underbrace{L_1 \times |\Delta z|}_{=0} + k_1 = k_1 \\
\boxed{\Delta z_2} &= L_2 \times |\Delta z| + k_2 \\
&= L_{2,1} \times \boxed{|\Delta z_1|} + k_2 \\
\boxed{\Delta z_3} &= L_3 \times |\Delta z| + k_3 \\
&= L_{3,1} \times \boxed{|\Delta z_1|} + L_{3,2} \times \boxed{|\Delta z_2|} + k_3 \\
\boxed{\Delta z_4} &= L_4 \times |\Delta z| + k_4 \\
&= L_{4,1} \times \boxed{|\Delta z_1|} + L_{4,2} \times \boxed{|\Delta z_2|} + L_{4,3} \times \boxed{|\Delta z_3|} + k_4 \\
....
\end{aligned}$$

9.2 Implementation

Our implementation is highly focused on speed and demands the device to hold all the required data structures in global memory simultaneously.

Given this premise, the calculation of (1) and (2) is a series of dot products and therefore is highly parallelize-able. For this we relied mainly on CUBLAS routines. The implementation is available on [...] with several interfaces in [...].

9.2.1 Performance

For measuring performance and the subsequent analysis, we simplified the process by equalizing the dimensions of the datasstuctures:

$$m = n = s$$

9.2.2 Single Execution

code

9.2.3 Multiple Executions

code

- one iteration with memory
- multiple executions

9.2.4 Analysis

- Memory Complexity $O(s^2)$
- Complexity $O(s^2)$
- mempry is bottle neck
- no performance gain expected

9.2.5 Notes

- Double Precision on GTX is nuts

10 Gradient

10.1 Problem Specification

ABS-Normal Form:

$$\begin{pmatrix} \Delta z \\ \Delta y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} Z & L \\ J & Y \end{pmatrix} \times \begin{pmatrix} \Delta x \\ |\Delta z| \end{pmatrix}$$

10.2 Implementation

10.3 Performance

10.4 Analysis

10.5 Notes

11 Operations

Operation	Function
Matrix - Vector Product	cublas
Matrix - Vector Vector Product	cublas
Vector Vector Addition	cuutils