GPU-Based Regression Analysis on Sparse Grids

Steffen Hirschmann

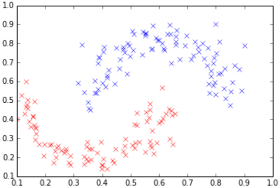
September 24, 2014

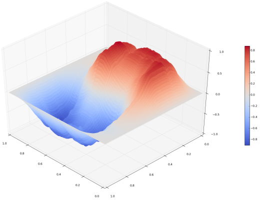
Regression

General Aspects

- Feature space Ω d-dimensional
- Assign a label $k \in \mathbb{R}$ to $\vec{x} \in \Omega$
- Supervised learning. Pre-classified data set:

$$S = \{(\vec{x}_i, y_i) \in \Omega \times \mathbb{R}\}_{i=1}^{m}$$





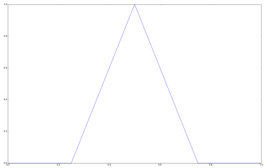
Regression

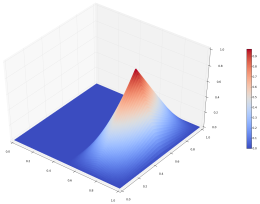
Approach on Sparse Grids

- Discretize Ω using sparse grids
- Piecewise d-linear sparse grid function

$$u(\vec{x}) = \sum_{i=1}^{g} v_i \phi_i(\vec{x})$$

- $\phi_i(\vec{x})$ tensor products of 1d hat functions.





Regression

As Least Squares Problem

- Determine u via a least squares problem

$$u = \underset{u}{\operatorname{argmin}} \left(\frac{1}{m} \sum_{i=1}^{m} (y_i - u(\vec{x_i}))^2 + \lambda \sum_{i=1}^{g} v_i^2 \right)$$

- Minimization by setting $\frac{\partial}{\partial v_i}$ to zero.
- Resulting SLE:

$$\left(\frac{1}{m} B B^T + \lambda I \right) \vec{v} = \frac{1}{m} B \vec{y}$$

- $B_{ij} = \phi_i(\vec{x_j})$
- Solve via CG; handled by framework SG^{++}

Implementation

Basic Algorithm - Parallelization

$$\underbrace{\begin{pmatrix} \phi_1(\vec{x_1}) & \phi_1(\vec{x_2}) & \dots & \phi_1(\vec{x_m}) \\ \phi_2(\vec{x_1}) & \phi_2(\vec{x_2}) & \dots & \phi_2(\vec{x_m}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_g(\vec{x_1}) & \phi_g(\vec{x_2}) & \dots & \phi_g(\vec{x_m}) \end{pmatrix}}_{B} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_g \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} \phi_1(\vec{x_1}) & \phi_2(\vec{x_1}) & \dots & \phi_g(\vec{x_1}) \\ \phi_1(\vec{x_2}) & \phi_2(\vec{x_2}) & \dots & \phi_g(\vec{x_2}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\vec{x_m}) & \phi_2(\vec{x_m}) & \dots & \phi_g(\vec{x_m}) \end{pmatrix}}_{B^T} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_g \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$

Implementation

Basic Algorithm - Parallelization

$$\underbrace{\left(\begin{array}{cccc} \phi_1(\vec{x_1}) & \phi_1(\vec{x_2}) & \dots & \phi_1(\vec{x_m}) \\ \hline \phi_2(\vec{x_1}) & \phi_2(\vec{x_2}) & \dots & \phi_2(\vec{x_m}) \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \phi_g(\vec{x_1}) & \phi_g(\vec{x_2}) & \dots & \phi_g(\vec{x_m}) \end{array}\right)}_{B} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix} = \left(\begin{array}{c} u_1 \\ \hline u_2 \\ \hline \vdots \\ \hline u_g \end{array}\right)$$

$$\underbrace{\left(\begin{array}{cccc} \phi_1(\vec{x_1}) & \phi_2(\vec{x_1}) & \dots & \phi_g(\vec{x_1}) \\ \hline \phi_1(\vec{x_2}) & \phi_2(\vec{x_2}) & \dots & \phi_g(\vec{x_2}) \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \phi_1(\vec{x_m}) & \phi_2(\vec{x_m}) & \dots & \phi_g(\vec{x_m}) \end{array}\right)}_{B^T} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_g \end{pmatrix} = \left(\begin{array}{c} v_1 \\ \hline v_2 \\ \hline \vdots \\ \hline v_m \end{array}\right)$$

Implementation

Basic Algorithm – Problems

Workload of one thread (index t):

Operation B:

$$\langle B_{t,\cdot}, v\rangle = \sum_{j=1}^{m} v_j \phi_t(\vec{x_j})$$

Operation B^T:

$$\langle B_{\cdot,t}, v\rangle = \sum_{j=1}^{g} \alpha_j \phi_j(\vec{x_t})$$

Implementation

Basic Algorithm – Problems

Workload of one thread (index t):

Operation B:

$$\langle B_{t,\cdot}, v\rangle = \sum_{j=1}^{m} v_j \prod_{d} \phi^{1\mathrm{D}}_{l_{t,d}, i_{t,d}}(x_{j,d})$$

Operation B^T:

$$\langle B_{\cdot,t}, v\rangle = \sum_{j=1}^{g} \alpha_j \prod_{d} \phi^{1\mathrm{D}}_{l_{j,d}, i_{j,d}}(x_{t,d})$$

Implementation

Basic Algorithm – Problems

Workload of one thread (index t):

Operation B:

$$\langle B_{t,\cdot}, v\rangle = \sum_{j=1}^{m} v_j \prod_d \phi^{1\text{D}}_{l_{t,d}, i_{t,d}}(x_{j,d})$$

Operation B^T:

$$\langle B_{\cdot,t}, v\rangle = \sum_{j=1}^{g} \alpha_j \prod_d \phi^{1\text{D}}_{l_{j,d}, i_{j,d}}(x_{t,d})$$

- Different number of addends ($\rightarrow$ deferred to memory management)

Implementation

Basic Algorithm – Problems

Workload of one thread (index t):

Operation B:

$$\langle B_{t,\cdot}, v\rangle = \sum_{j=1}^{m} v_j \prod_d \phi^{1D}_{l_{t,d}, i_{t,d}}(x_{j,d})$$

Operation B^T:

$$\langle B_{\cdot,t}, v\rangle = \sum_{j=1}^{g} \alpha_j \prod_d \phi^{1D}_{l_{j,d}, i_{j,d}}(x_{t,d})$$

- Different number of addends ($\rightarrow$ deferred to memory management)
- Small inner loop

Implementation

Basic Algorithm – Problems

Workload of one thread (index t):

Operation B:

$$\langle B_{t,\cdot}, v\rangle = \sum_{j=1}^{m} v_j \prod_d \phi^{1D}_{l_{t,d},i_{t,d}}(x_{j,d})$$

Operation B^T:

$$\langle B_{\cdot,t}, v\rangle = \sum_{j=1}^{g} \alpha_j \prod_d \phi^{1D}_{l_{j,d},i_{j,d}}(x_{t,d})$$

- Different number of addends ($\rightarrow$ deferred to memory management)
- Small inner loop
- Thread cannot keep associated values in registers

Implementation

Basic Algorithm – Problems

Workload of one thread (index t):

Operation B:

$$\langle B_{t,\cdot}, v \rangle = \sum_{j=1}^{m} v_j \prod_{d} \phi^{1\mathrm{D}}_{l_{t,d}, i_{t,d}}(x_{j,d})$$

Operation B^T:

$$\langle B_{\cdot,t}, v \rangle = \sum_{j=1}^{g} \alpha_j \prod_{d} \phi^{1\mathrm{D}}_{l_{j,d}, i_{j,d}}(x_{t,d})$$

- Different number of addends ($\rightarrow$ deferred to memory management)
- Small inner loop
- Thread cannot keep associated values in registers
- Bad access pattern to global memory

Implementation

Compile Time Constant Dimension

$//\ldots$
for $d \leftarrow 0..D-1$ **do**
$\quad p \leftarrow p \cdot \phi^{1\mathrm{D}}_{l_{t,d},i_{t,d}}(x_{j,d})$
end for
$//\ldots$

Implementation

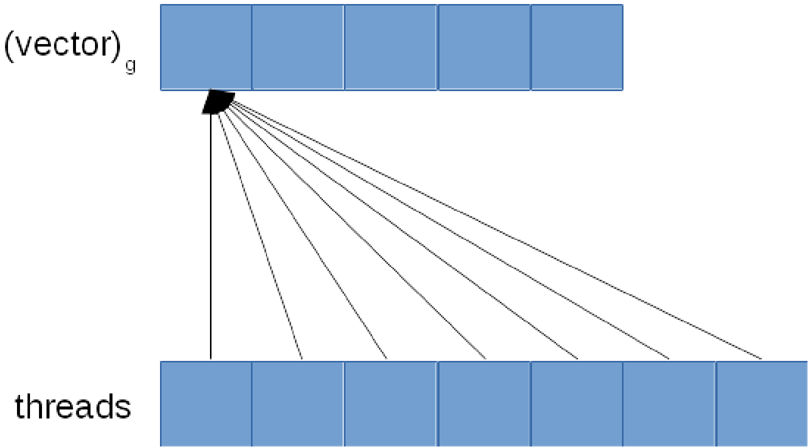
Compile Time Constant Dimension

template $\langle$**int** $D\rangle$

$$
\begin{array}{l}
//\ldots \\
\textbf{for } d \leftarrow 0..D-1 \textbf{ do} \\
\quad p \leftarrow p \cdot \phi^{1\mathrm{D}}_{l_{t,d}, i_{t,d}}(x_{j,d}) \\
\textbf{end for} \\
//\ldots
\end{array}
\quad \overset{\textit{Compiler}}{\Longrightarrow} \quad
\begin{array}{l}
//\ldots \\
p \leftarrow p \cdot \phi^{1\mathrm{D}}_{l_{t,0}, i_{t,0}}(x_{j,0}) \\
\qquad \vdots \\
p \leftarrow p \cdot \phi^{1\mathrm{D}}_{l_{t,D-1}, i_{t,D-1}}(x_{j,D-1}) \\
//\ldots
\end{array}
$$

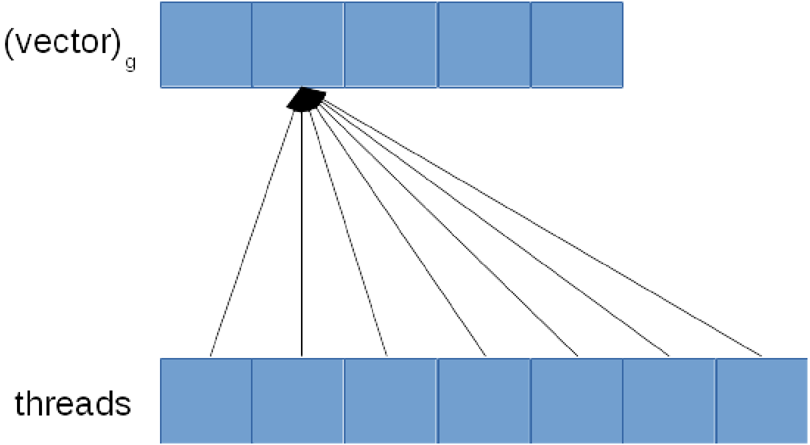
Implementation

Shared Memory



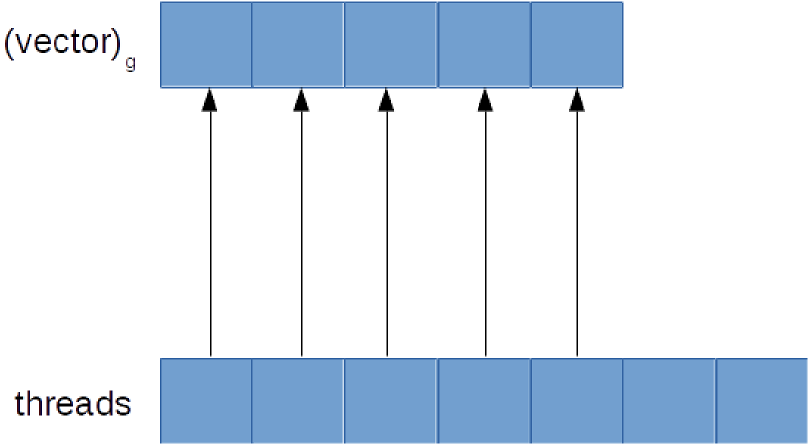
Implementation

Shared Memory



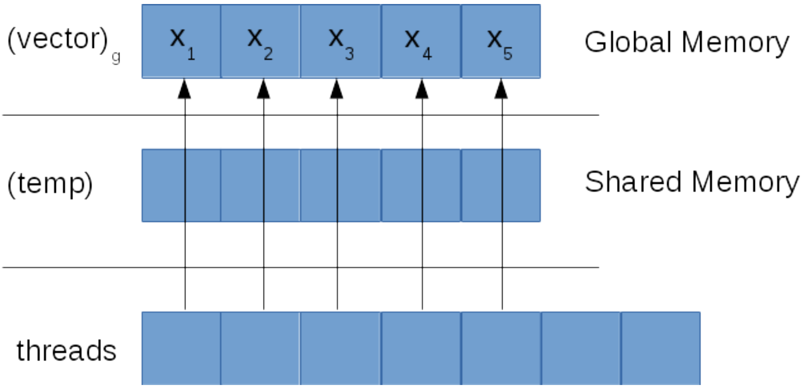
Implementation

Shared Memory



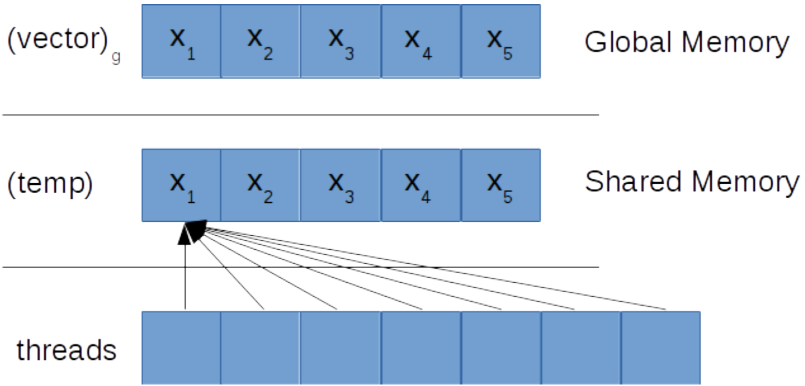
Implementation

Shared Memory



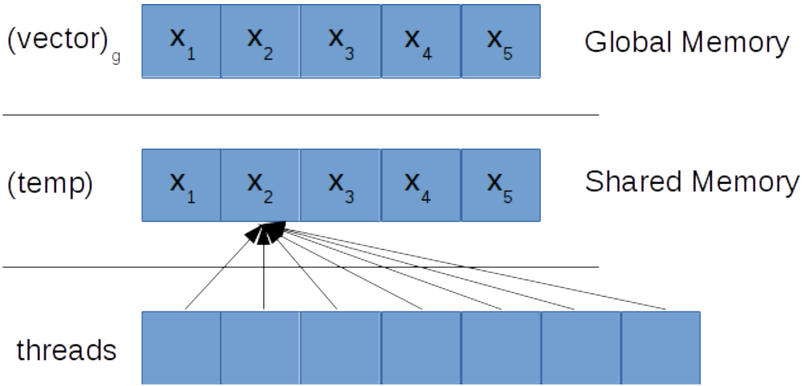
Implementation

Shared Memory



Implementation

Shared Memory



Memory Management

The necessary data:

$$(l_{k,l}) \in \mathbb{R}^{g \times d}, (i_{k,l}) \in \mathbb{R}^{g \times d}, (x_{i,d}) \in \mathbb{R}^{m \times d}$$

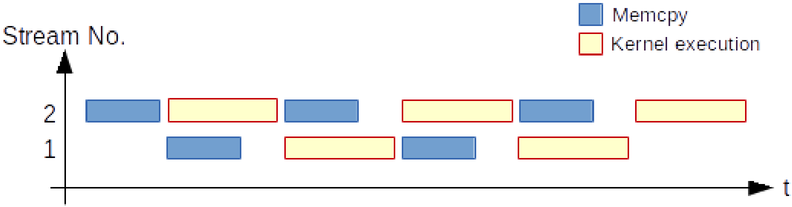
Memory Management Problems:

- $m \gg g$ (Different number of addends)
- Program spends much time simply copying data
- Data might not fit entirely into device memory

Memory Management

Strategy:

1. Divide dataset into N chunks
2. Start $n < N$ streams and schedule the N chunks



Results

- Single precision!
- DR5: Real-world dataset $\sim$ 40 MB
- Intel Core i7 2600: **468.269 s**
- NVIDIA GeForce GTX680:

Templates	Streams	Shared Mem	Timing
✗	✗	✗	812.242 s
✓	✗	✗	156.367 s
✓	✓	✗	94.2451 s
✓	✓	✓	**77.9071 s**

Summary

- Discretization via Sparse Grids
- Least-Squares-Problem; minimization yields SLE
- Important techniques:
  - Allow the compiler to use registers
  - Unroll inner loops
  - Data access patterns + memory hierarchy
  - Balance operations
  - Streaming