GPU-Based Regression Analysis on Sparse Grids

Steffen Hirschmann

September 24, 2014
Regression

General Aspects

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

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

![Graph](image1.png)

![Graph](image2.png)
Regression
Approach on Sparse Grids

- Discretize $\Omega$ 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

\[
 u = \arg\min_u \left( \frac{1}{m} \sum_{i=1}^{m} (y_i - u(\vec{x}_i))^2 + \lambda \sum_{i=1}^{g} v_i^2 \right)
\]

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

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

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

Steffen Hirschmann
GPU-Based Regression Analysis on Sparse Grids
SKILL 2014
Implementation

Basic Algorithm - Parallelization

\[
\begin{pmatrix}
\phi_1(x_1) & \phi_1(x_2) & \ldots & \phi_1(x_m) \\
\phi_2(x_1) & \phi_2(x_2) & \ldots & \phi_2(x_m) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_g(x_1) & \phi_g(x_2) & \ldots & \phi_g(x_m)
\end{pmatrix}
\begin{pmatrix}
v_1 \\
v_2 \\
\vdots \\
v_m
\end{pmatrix}
=
\begin{pmatrix}
u_1 \\
u_2 \\
\vdots \\
u_g
\end{pmatrix}
\]

\[
\begin{pmatrix}
\phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_g(x_1) \\
\phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_g(x_2) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_1(x_m) & \phi_2(x_m) & \ldots & \phi_g(x_m)
\end{pmatrix}
\begin{pmatrix}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_g
\end{pmatrix}
=
\begin{pmatrix}
v_1 \\
v_2 \\
\vdots \\
v_m
\end{pmatrix}
\]
Implementation

Basic Algorithm - Parallelization

\[
\begin{pmatrix}
\phi_1(\vec{x}_1) & \phi_1(\vec{x}_2) & \ldots & \phi_1(\vec{x}_m) \\
\phi_2(\vec{x}_1) & \phi_2(\vec{x}_2) & \ldots & \phi_2(\vec{x}_m) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_g(\vec{x}_1) & \phi_g(\vec{x}_2) & \ldots & \phi_g(\vec{x}_m)
\end{pmatrix}
\begin{pmatrix}
v_1 \\
v_2 \\
\vdots \\
v_m
\end{pmatrix}
=
\begin{pmatrix}
u_1 \\
u_2 \\
\vdots \\
u_g
\end{pmatrix}
\]

\[
\begin{pmatrix}
\phi_1(\vec{x}_1) & \phi_2(\vec{x}_1) & \ldots & \phi_g(\vec{x}_1) \\
\phi_1(\vec{x}_2) & \phi_2(\vec{x}_2) & \ldots & \phi_g(\vec{x}_2) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_1(\vec{x}_m) & \phi_2(\vec{x}_m) & \ldots & \phi_g(\vec{x}_m)
\end{pmatrix}
\begin{pmatrix}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_g
\end{pmatrix}
=
\begin{pmatrix}
v_1 \\
v_2 \\
\vdots \\
v_m
\end{pmatrix}
\]
Workload of one thread (index $t$):

Operation $B$:

$$\langle B_t, v \rangle = \sum_{j=1}^{m} v_j \phi_t(\vec{x}_j)$$

Operation $B^T$:

$$\langle B^T, v \rangle = \sum_{j=1}^{g} \alpha_j \phi_j(\vec{x}_t)$$
Implementation

Basic Algorithm – Problems

Workload of one thread (index $t$):

Operation $B$:

\[
\langle B_{t, \cdot}, v \rangle = \sum_{j=1}^{m} v_j \prod_{d} \phi_{l_{t,d}, i_{t,d}}^{1D}(x_j, d)
\]

Operation $B^T$:

\[
\langle B_{\cdot, t}, v \rangle = \sum_{j=1}^{g} \alpha_j \prod_{d} \phi_{l_{j,d}, i_{j,d}}^{1D}(x_t, d)
\]
Implementation
Basic Algorithm – Problems

Workload of one thread (index \( t \)):

Operation \( B \):

\[
\langle B_{t,:), v \rangle = \sum_{j=1}^{m} v_j \prod_d \phi_{l,t,d,i_t,d}^{1D}(x_j,d)
\]

Operation \( B^T \):

\[
\langle B_{:,t}, v \rangle = \sum_{j=1}^{g} \alpha_j \prod_d \phi_{l,j,d,i_j,d}^{1D}(x_t,d)
\]

- Different number of addends (\( \rightarrow \) deferred to memory management)
Workload of one thread (index $t$):

**Operation $B$:**

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

**Operation $B_T$:**

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

- Different number of addends (→ deferred to memory management)
- Small inner loop
Implementation

Basic Algorithm – Problems

Workload of one thread (index $t$):

Operation $B$:

$$\langle B_{t,.}, v \rangle = \sum_{j=1}^{m} v_j \prod_{d} \phi_{l_t,d,i_t,d}^{1D}(x_{j,d})$$

Operation $B^T$:

$$\langle B_{.,t}, v \rangle = \sum_{j=1}^{g} \alpha_j \prod_{d} \phi_{l_t,d,i_t,d}^{1D}(x_{t,d})$$

- 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 $B$:

$$
\langle B_{t, \cdot}, v \rangle = \sum_{j=1}^{m} v_j \prod_{d} \phi^{1D}_{l_t, d, i_t, d}(x_j, d)
$$

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 (→ deferred to memory management)
- Small inner loop
- Thread cannot keep associated values in registers
- Bad access pattern to global memory
//... for d ← 0..D − 1 do
p ← p · φ_{l_t,d,i_t,d}^{1D}(x_j,d)
end for
//...
Implementation
Compile Time Constant Dimension

\[
\text{template } \langle \text{int } D \rangle
\]

//...
for \( d \leftarrow 0..D - 1 \) do
\[
p \leftarrow p \cdot \phi_{l_t,d,i_t,d}^{1\text{D}}(x_j,d)
\]
end for
//...

//...
\[
p \leftarrow p \cdot \phi_{l_t,0,i_t,0}^{1\text{D}}(x_j,0)
\]

\[
\Rightarrow
\]
Compiler

\[
p \leftarrow p \cdot \phi_{l_t,D-1,i_t,D-1}^{1\text{D}}(x_j,D-1)
\]
//...

//...
Implementation
Shared Memory

(vector) \_g

threads
Implementation

Shared Memory

(vector)\_g

threads
Implementation

Shared Memory

(vector)_{g, \text{threads}}
Implementation

Shared Memory

(vector)\_g

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \]

Global Memory

(temp)

Shared Memory

threads
Implementation

Steffen Hirschmann

GPU-Based Regression Analysis on Sparse Grids

SKILL 2014
Implementation

Shared Memory

\((\text{vector})_g\)

Global Memory

\((\text{temp})\)

Shared Memory

threads
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 \( \sim 40 \text{ MB} \)
- Intel Core i7 2600: **468.269 s**
- NVIDIA GeForce GTX680:

<table>
<thead>
<tr>
<th>Templates</th>
<th>Streams</th>
<th>Shared Mem</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\times)</td>
<td>(\times)</td>
<td>(\times)</td>
<td>812.242 s</td>
</tr>
<tr>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
<td>156.367 s</td>
</tr>
<tr>
<td>(\checkmark)</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>94.2451 s</td>
</tr>
<tr>
<td>(\checkmark)</td>
<td>(\checkmark)</td>
<td>(\checkmark)</td>
<td><strong>77.9071 s</strong></td>
</tr>
</tbody>
</table>
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