fastsg: A Fast Routines Library for Sparse Grids

Alin Murarasu (murarasu@in.tum.de), Gerrit Buse, Dirk Pflüger, Josef Weidendorfer, Arndt Bode

Technische Universität München
Overview

- Compression / Decompression of Simulation Data
- Sparse Grids
- fastsg's Interface
- Data Structure
- Sparse Grid Algorithms
- Performance
High-dimensional Data Exploration

- Simulations on supercomputers
  => large-scale data
  => high-dimensional data
     - e.g. # of dims. ~= 10

- We want to visualize / interact with the data
- But data is too big for visualization nodes
  => we need a compressed / hierarchical data representation
Sparse grids (d-dimensional) != sparse matrices (2d)

- The sparse grid technique
  - defines hierarchical basis fcts.
  - excludes the basis fcts. with reduced contribution

- Sparse grids require fewer grid points / values
  - $O(N^*(\log(N))^{d-1})$ vs. $O(N^d)$ for full grids
  - error only slightly deteriorated for smooth fcts.
  - => compression (lossy)
1d Hierarchical Basis Functions

- Hierarchy of basis functions
- Incremental levels of detail
- Level ++ => support --
- Each basis fct. is identified by \((l, i)\)
  - \(l = \text{level}\)
  - \(i = \text{index}\)
- Sparse grid approximation = basis fcts. + coefficients
- 2d basis fct. = product of two 1d basis fcts.
- Each grid point
  - contains a coefficient that scales a basis fct.
  - is identified by \((l, i)\) (\(l\) and \(i\) are now vectors)
- Approximation = sum of scaled basis fcts.
Building Sparse Grids

- Assumption: support $\sim$ contribution
- Central idea: exclude basis fcts. with the smallest support (they count less)
- How? By restricting the L1-norm of $l = (l_1, l_2)$
  - e.g. $l_1 + l_2 \leq n$

Constraint: $l_1 + l_2 \leq 4$
Dimensionally Truncated Sparse Grids

- Regular sparse grids (left):
  - same # of points per dimension
  - one constraint: \( l_1 + \ldots + l_d \leq n \)

- Dimensionally truncated sparse grids (right):
  - different # of points for different dimensions
  - \( d + 1 \) constraints: \( l_1 + \ldots + l_d \leq n, l_1 \leq c_1, \ldots, l_d \leq c_d \)

- Why? different dimensions might need different levels of detail
fastsg's Interface

- **Dual-layer:**
  - **data structure** with minimal memory footprint (no trees, hash-tables, …)
  - **high-level operations** optimized for caches and vector units

- **User-friendly:**
  1. **Initialize:** inserts the values of a fct. \( f \) in the sparse grid
  2. **Access data:** \( \text{agp2idx} \) returns an index used to get the value at \((l, i)\)
  3. **Hierarchize:** computes the hierarchical coefficients
  4. **Evaluate:** interpolates the sparse grid

<table>
<thead>
<tr>
<th>Layer</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structure</td>
<td>\text{init}(d, n, c[d], f)</td>
</tr>
<tr>
<td></td>
<td>\text{agp2idx}(l[d], i[d])</td>
</tr>
<tr>
<td></td>
<td>\text{idx2agp}(idx)</td>
</tr>
<tr>
<td></td>
<td>\text{next}(l[d])</td>
</tr>
<tr>
<td></td>
<td>\text{size}()</td>
</tr>
<tr>
<td>Sparse grid operations</td>
<td>\text{hierarchize}()</td>
</tr>
<tr>
<td></td>
<td>\text{evaluate}(x[m][d])</td>
</tr>
<tr>
<td></td>
<td>\text{error}()</td>
</tr>
</tbody>
</table>
## Storage Scheme

**Central idea:**
- decompose the sparse grid in dense blocks
- **Only the values are stored in a special order**
  => minimal memory consumption

**To return the value at** \((l, i)\), `agp2idx` adds 3 indices:
- \(idx_1\): beginning of the group that contains \((l, i)\)
- \(idx_2\): beginning of the block that contains \((l, i)\)
- \(idx_3\): position of the value in the block

**Before indexing, there are 2 questions:**
- \(P_1\): what is the size of a group?
- \(P_2\): what is the position of a block in its group?

**Trade-off:** int ops ++, mem. refs. --
Dynamic Programming Algorithms

- fastsg includes dynamic programming algorithms for \( P_1 \) and \( P_2 \)

- \( P_1 \): what is the size of a group?
  - let \( A(i, j) \) be the # of vectors with length \( i \) and L1-norm \( j \)
    - if \( j < c \), then \( A(i, j) = A(i, j - 1) + A(i - 1, j) \)
    - else \( A(i, j) = A(i, j - 1) + A(i - 1, j - c) \)
  - our algorithm fills the matrix \( A(d, n) \) in \( O(d \times n) \) time
  - \( A \) is computed once at initialization

- \( P_2 \): what is the position of a block in its group?
  - we define an order / sequence for vectors \( l \) with the same L1-norm
    - the sequence is implicit, i.e. not stored in memory
  - we determine the position of any \( l \) in the sequence in \( O(d + n) \) time
  - this operation is used intensively
Sparse Grid Algorithms

- Hierarchization / compression:
  - indirect accesses to memory => memory bound
  - \textit{idx2agp / agp2idx} are expensive => integer bound

- Evaluation / interpolation / decompression:
  - we evaluate the sparse grid at \( m \) points, \( m \geq 10,000 \) => CPU bound

Optimization philosophy = exploit dense blocks
- broadcast a micro-operation to an entire block (SIMD-like)
- allows for better use of caches / vector units
Sequential Performance

- **Hierarchization (< 5x):**
  - opt1, opt2, opt3: loop invariant code motion
  - opt4: vectorization
- **Evaluation (< 7x):**
  - opt1: loop interchange
  - opt2: vectorization
- **Following the same path, our most recent results are:**
  - hierarchization: 1.3 Gflops (< 26x)
  - evaluation: 8.4 Gflops (< 12x)
Evaluation benefits from Hyperthreading
- there is optimization potential for improving serial performance
- important for systems without multi-threading, e.g. AMD

This behavior is visible with gcc, not icc

We managed to help the gcc compiler using manual unroll-and-jam
=> more ILP in the innermost loop
=> same performance as icc
Conclusion

- fastsg provides fast routines for the sparse grid technique
- Its focus is on approximation / interpolation
- It supports regular and dimensionally truncated sparse grids
- Its features include:
  - minimal memory consumption
  - optimized high-level operations for compression functionality
  - simple to use interface
Thank you for listening!