BLIS-Based High Performance Computing Kernels in N-body Problems

Chenhan D. Yu

The 3rd BLIS Retreat
Sep 28, 2015
N-body Problems
N-body Problems

- N-body problems aim to describe the interaction (relation) of \( N \) points \( \{ \mathbf{X} \} \) in a \( d \) dimensional space.
- \( K(x_i, x_j) = K_{ij} \) describes the interaction between \( x_i \) and \( x_j \).
- 3 operations: Kernel Summation \( u = Kw \), Kernel Inversion \( w = (K + \lambda I)^{-1}u \) and Nearest-Neighbors.
- 2D and 3D applications can be found in computational physics, geophysical exploration and medical imaging.
- High dimension applications in computational statistic include clustering, classification and regression.
Outline

- Kernel Summation \((u=Kw)\) and Nearest-Neighbors.

- How \texttt{GEMM} is applied in the conventional approach?

- Why \texttt{GEMM} can be memory bound in these operations?

- What insight is required to design an algorithm that avoids redundant memory operations but still preserves the efficiency?

- How \texttt{GSKS} and \texttt{GSKNN} are inspired by the \texttt{BLIS} framework in their design?
### Linear Kernel

The linear kernel is defined as:

$$K(x_i, x_j) = x_i^T x_j$$

#### Example

Let's consider the following points:

- $x_1 = (0, 1)$
- $x_2 = (1, 0)$
- $x_3 = (1, 1)$

The kernel matrix $K$ can be computed as:

$$K = Q^T R$$

where $Q$ and $R$ are matrices resulting from the transformation and kernel summation, respectively.

#### Kernel Summation

$$K_w = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

#### Nearest-Neighbors

$$x_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$K_w$$

$$x_2 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

$$x_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Copyright © 2015, The University of Texas at Austin
Other Kernels

\[ K(x_i, x_j) = f(x_i^T x_j), \text{ e.g. Gaussian kernel} \]

\[ K(x_i, x_j) = \exp(-\|x_i - x_j\|^2_2/(2h^2)) \]

The expansion exposes \textbf{GEMM} operations:

\[ \|x_i - x_j\|^2_2 = \|x_i\|^2_2 + \|x_j\|^2_2 - 2x_i^T x_j \]
The Big Picture

- $Kw$ takes $O(N^2)$ if $K$ is precomputed, otherwise $O(dN^2)$. The cost is too expensive when $N$ is large.

- Exhaustive search requires $O(N^2 \log(k))$ if $K$ is precomputed, otherwise $O(dN^2+N^2 \log(k))$.

- Divide-and-conquer approximation: Barnes-Hut or FMM for kernel summation, and randomized KD-tree or locality sensitive hashing for kNN.

- Still the subproblem of all these algorithms is to solve several smaller dense kernel summation or kNN.

- Solving the subproblem fast benefits all these methods.
Subproblem

- Take two subsets $Q$ and $R$ from $X$.
- Compute $K(Q, R)$ with GEMM using:
  \[
  \left\| x_i - x_j \right\|_2^2 = \left\| x_i \right\|_2^2 + \left\| x_j \right\|_2^2 - 2 x_i^T x_j
  \]
- Compute $Kw$ with GEMV or select $k$ entries in each row.
- Rely on BLAS, VML (Vectorized Math Library) and STL.
- What can possibly go wrong?
Visualization

\[ N \times d = n \times x \times d = m \times K \]

Copyright © 2015, The University of Texas at Austin
Insights

- $Q$, $R$ and $K$ can’t be stored.
- Collect $Q$ and $R$ from $X$ during packing.
- $K(x_i, x_j) = K_{ij}$ must be computed in registers.
- $K_w$ or $k$-select must be completed in registers.
- Only store the output.
- We need a special packing routine.
- Fuse GEMM with distance calculations, special function evaluations, $K_w$ or $k$-select.

i.e.
Code Fusion in BLIS

Code fusion is done in micro-kernel, and the BLIS framework is maintained.

Copyright © 2015, The University of Texas at Austin
GSKNN and BLIS ($K=Q^TR$)
**Micro-Kernel**

```
LOAD Q
LOAD R
FMA Q, R, C03_0
SHUFFLE Q, R, C03_1
FMA Q, R, C03_2
PERMUTE2F128 Q, R, C03_3
FMA Q, R, C03_3
SHUFFLE Q, R, C03_3
FMA Q, R, C03_3
```

**Micro-Kernel**

- **LOAD**
  - Load Q
  - Load R

- **FMA**
  - FMA Q, R, C03_0
  - FMA Q, R, C03_1

- **PERMUTE2F128**
  - Permute Q, R, C03_2

- **SHUFFLE**
  - Shuffle Q, R, C03_3
Micro-Kernel

LOAD
LOAD
FMA
SHUFFLE
FMA
PERMUTE2F128
FMA
SHUFFLE
FMA

LOAD Q
LOAD R
FMA Q, R, C03_0
SHUFFLE FMA Q, R, C03_1
PERMUTE2F128 FMA Q, R, C03_2
SHUFFLE FMA Q, R, C03_3

R1 R0 R2 R3
Q0 00 01 03
Q1 10 11 12
Q2 21 22 23
Q3 30 32 33
LOAD Q
LOAD R
FMA Q, R, C03_0
SHUFFLE FMA Q, R, C03_1
PERMUTE2F128 FMA Q, R, C03_2
SHUFFLE FMA Q, R, C03_3
Micro-Kernel with p-norm

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Q, R, C03_0</th>
<th>Q, R, C03_1</th>
<th>Q, R, C03_2</th>
<th>Q, R, C03_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHUFFLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERMUTE2F128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHUFFLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1-norm
- SUB
- AND (flip signed bit)
- ADD

inf-norm
- SUB
- AND (flip signed bit)
- MAX

p-norm
- SUB
- POW (SVML)
- ADD
With a high precision (20 digits in decimal), **Remez exchange algorithm** can generate an 11 order near minimax polynomial with $1E-18$ relative error.

$$P_{11}(x) = c_{11} + (\ldots + (c_5 + (c_4 + (c_3 + (c_2 + (c_1 + c_0 x)x)x)x)x)x\ldots)x$$

$$= 1\text{ADD} + 11\text{FMA}$$
Vectorized Max Heap

Find the max child

C: [1, 3, 4, 2] -> [4, 3, 4, 3] -> [4, 4, 4, 4]  
D: [4, 2, 1, 3]  [3, 4, 3, 4]

LOAD         C
SHUFFLE      D, C, 0x5
MAX          D, C
PERMUTE2F128 C, D, 0x1
MAX          D, C

Copyright © 2015, The University of Texas at Austin
Efficiency Analysis

\[ T_{BLAS} = T_{GSKS} + T_R + T_Q + T_K \]

\[ \frac{mn(2d+36)}{T_{GSKS}} - \frac{mn(2d+36)}{T_{BLAS}} = ? \]
In this section we give details on the experimental setup.

**Memory Bound Efficiency Graphs**

- **Variant#1, m=n=8192, k=16**
- **Variant#1, m=n=8192, k=512**
- **#1, nthd=10, m=n=8192, k=16**
- **#1, nthd=10, m=n=8192, k=512**

Copyright © 2015, The University of Texas at Austin
Conclusion

- The **GEMM** approach in N-body problems is a good example to show the current BLAS library is lacking flexibility for lower level integration.

- The algorithmic innovation of **GSKS** and **GSKNN** is to break through the interface, seeking for the lowest memory complexity.

- We exploit these observations with the help of the **BLIS** framework.

- Ongoing work includes other operations. e.g. kernel inversion, **k**-meaning clustering. Port to GPU and other accelerators.
Question?

github.com/ChenhanYu/ks  
github.com/ChenhanYu/rnn