Parallelism in Linnea

Henrik Barthels, Paolo Bientinesi
Introduction: Linnea

• How to compute the following expressions?

\[ b := (X^T X)^{-1} X^T y \]
\[ b := (X^T M^{-1} X)^{-1} X^T M^{-1} y \]
\[ x := W(A^T (AWA^T)^{-1} b - c) \]
\[ x := (A^{-T} B^T BA^{-1} + R^T [\Lambda (Rz)] R)^{-1} A^{-T} B^T BA^{-1} y \]

• High-level languages are easy to use, but performance is usually suboptimal.
• BLAS and LAPACK can be fast, but require a lot of expertise.

BLAS [DDC+90], LAPACK [AB+99]

• \( y \leftarrow A x + y \)
• \( C \leftarrow A B + C \)
• \( B \leftarrow A^{-1} B \)
• \( \ldots \)
Introduction: Linnea

Input

\[ z := (X^T S^{-1} X)^{-1} X^T S^{-1} y \]  \quad S \text{ is symmetric positive definite}

Output

\begin{align*}
L &:= \text{chol}(S) \quad \text{(potrf)} \\
U_1 &:= L^{-1} X \quad \text{(trsm)} \\
(Q, R) &:= \text{qr}(U_1) \quad \text{(geqrf)} \\
u_2 &:= L^{-1} y \quad \text{(trsv)} \\
u_3 &:= Q^T u_2 \quad \text{(ormqr)} \\
z &:= R^{-1} u_3 \quad \text{(trsv)}
\end{align*}

https://github.com/HPAC/linnea
Linear Algebra Knowledge

- Properties: `trmm` vs. `gemm`
- Inference of properties: \[ \begin{bmatrix} A \\ B \end{bmatrix} \rightarrow AB \]
- Simplifications: \( A^T \rightarrow A \) if `Symmetric(A)`
- Rewriting expressions:
  \[
  X := A^T A + A^T B + B^T A \quad \rightarrow \quad Y := B + A/2 \\
  X := A^T Y + Y^T A
  \]
- Common subexpressions:
  \[
  X := AB^{-T} C + B^{-1} A^T \quad \rightarrow \quad Z := AB^{-T} \\
  X := ZC + Z^T
  \]
- Matrix chains:
  \[
  (AB)c \quad \mathcal{O}(n^3) \\
  A(BC) \quad \mathcal{O}(n^2)
  \]
Problem 1: Generation of Parallel Code

Problem 2: Parallel Code Generation
Problem 1: Generation of Parallel Code

Problem 2: Parallel Code Generation
Generation of Parallel Code

How to make use of parallelism?

• Threaded kernels.
• Kernels in parallel.
• Both.
Generation of Parallel Code

Step 1: Parallelize a given sequence of kernels.

The Good

- Constructing dependency graph is easy.

\[
T_1 := AB \\
T_2 := L^{-1}C \\
X := T_1 + T_2
\]

- If operand sizes are known, amount of work is known (#FLOPs).

\[
A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times n} \rightarrow AB \text{ requires } 2mnk \text{ FLOPs}
\]

- BLAS and LAPACK are parallelized.
The Bad

- FLOP count is not a good prediction for execution time.
- Performance modeling is a hard problem [PB12].
  - Performance is not composable.
  - Efficiency decreases with number of threads.

![Graph showing efficiency of GEMM vs matrix dimension](image)
Generation of Parallel Code

The Bad

- FLOP count is not a good prediction for execution time.
- Performance modeling is a hard problem [PB12].
  - Performance is not composable.
  - Efficiency decreases with number of threads.
  - “Overbooking”: Parallelizing a sequence of $n$ LU's [PB15].
Generation of Parallel Code

Existing Tools

PaRSEC, OmpSs, StarPU, SuperGlue,…

• Built for large dependency graphs/large number of tasks.
• Only one thread per task.
• Do not consider cost (except StarPU [ATN09]).

We have:

• Small number of tasks.
• Multiple threads per task.
• Cost is (roughly) known.
Generation of Parallel Code

Step 2: Generate sequence that parallelizes well.

A good sequence of kernels for sequential execution may not be good for parallel execution.

Example: $X := ABCD$

$A, B, C \in \mathbb{R}^{n \times n}$

$D \in \mathbb{R}^{n \times m}$

$m < n$

$A(B(CD))$

- min #FLOPs
- dependencies
- threaded kernels still possible

$(AB)(CD)$

- more FLOPs
- fewer dependencies

Existing work: Matrix Chain Products on Parallel Systems [LKHL03]
Problem 1: Generation of Parallel Code

Problem 2: Parallel Code Generation
Parallel Code Generation

Motivation

• For sufficiently large matrices and/or enough runs of the program, generation time will be amortized.
• What about small computations?
• What about computations in interactive environment such as Matlab?
Parallel Code Generation

\[ b := X^T M^{-1} X y \]
\[ a := X^T M^{-1} X y \]

\[ b := X^T (L^T L)^{-1} X y \]

\[ L := \text{chol}(M) \]

\[ b := X^T (L^T L)^{-1} X y \]

...
Parallel Code Generation

\[ L := \text{chol}(M) \]

\[ b := X^T M^{-1} X y \]

\[ b := X^T L^{-1} L^{-T} X y \]

\[ \ldots \]
Parallel Code Generation

\[ b := X^T M^{-1} X y \]

\[ L := \text{chol}(M) \]

\[ b := X^T L^{-1} L^{-T} X y \]

\[ Z := L^{-T} X \]

\[ b := Z^T Z y \]

\[ \ldots \]
Parallel Code Generation

\[
L := \text{chol}(M)
\]

\[
b := X^T M^{-1} X y
\]

\[
Z := L^{-T} X
\]

\[
b := Z^T Z y
\]

\[
t_1 := X y
\]

\[
b := X^T L^{-1} L^{-T} t_1
\]

\[
\ldots
\]
Parallel Code Generation

\[
L := \text{chol}(M)
\]

\[
b := X^T M^{-1} X y
\]

\[
Z := L^{-T} X
\]

\[
b := Z^T Z y
\]

\[
t_1 := X y
\]

\[
b := X^T L^{-1} L^{-T} t_1
\]

\[
\ldots
\]
Parallel Code Generation

Reducing Redundancy

\[ X := ABC + CDE \]

\[ T_1 := AB \]
\[ X := T_1 C + CDE \]
\[ T_2 := T_1 C \]
\[ X := T_2 + CDE \]

\[ T_3 := BC \]
\[ X := AT_3 + CDE \]
\[ T_4 := AT_3 \]
\[ X := T_4 + CDE \]
Parallel Code Generation

Reducing Redundancy

\[ X := ABC + CDE \]
\[ T_1 := AB \quad T_3 := BC \]
\[ X := T_1 C + CDE \]
\[ T_2 := T_1 C \quad T_2 := AT_3 \]
\[ X := T_2 + CDE \]

Table of expressions and intermediate operands:

<table>
<thead>
<tr>
<th>tmp</th>
<th>expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td>( AB )</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>( ABC )</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>( BC )</td>
</tr>
</tbody>
</table>
Reducing Redundancy

- exhaustive, merging
- exhaustive, no merging
- constructive, merging
- constructive, no merging
$z := (X^T S^{-1} X)^{-1} X^T S^{-1} y$  

S is symmetric positive definite
\( x := W(A^T(AWA^T)^{-1}b - c) \)

\( W \) is diagonal, diagonal elements are positive
Parallel Code Generation

The Good
• Finding redundant nodes can be done efficiently using hash table.

The Bad
• Access to table of expressions & intermediates has to be protected.
• Merging nodes requires synchronization.
• Graph is not uniform at all.
• Graph is initially not known.

Possible solutions:
• Tradeoff between merging and redundancy?
• Parallelize computations on nodes?
• Nodes as tasks?
References


References
