IOOpt: Automatic Derivation of I/O Complexity Bounds for Affine Programs

Auguste Olivry  Guillaume Iooss  Nicolas Tollenaere
Atanas Rountev  P. Sadayappan  Fabrice Rastello
June 2021
What is I/O complexity?

- Arithmetic complexity = # of operations

![Diagram of CPU, fast memory, and slow memory with capacity S and unbounded capacity.]
What is I/O complexity?

- Arithmetic complexity = # of operations
What is I/O complexity?

- Arithmetic complexity = # of operations

Diagram:
- CPU
- Fast memory capacity $S$
- Slow memory unbounded capacity
What is I/O complexity?

- Arithmetic complexity = # of operations
- I/O cost (schedule-dependent) = amount of data moved between fast and slow memory
What is I/O complexity?

- Arithmetic complexity = # of operations
- I/O cost (schedule-dependent) = amount of data moved between fast and slow memory
- I/O complexity = minimum cost over all schedules
Lower and Upper Bounds

\[ \text{LB} \leq \text{IO} \]

IOLB (PLDI '20)
Automated lower bound computation
IOLB (*PLDI ’20*)
Automated lower bound computation

**IOOpt** (This paper)

- Improvement of the lower bound algorithm
- Automated upper bound derivation (IOUB)
I/O complexity upper bound ⇔ Cost of a particular valid schedule

Untiled matrix multiplication

I/O cost: $O(N^3)$

Tiled matrix multiplication

I/O cost: $O\left(\frac{N^3}{\sqrt{S}}\right)$

→ How to automatically compute I/O cost for a given schedule?
Upper bound derivation

Input program

Pruning algorithm

Tiling loop permutations

For each permutation

Parametrically tiled program

Polyhedral calculus

Symbolic I/O cost expressions

Parameter values
Operations research
Computer algebra

Tile sizes
Bound as a function of $S$
Upper bound derivation

Input program

Pruning algorithm

Tiling loop permutations

For each permutation

Parametrically tiled program

Polyhedral calculus

Symbolic I/O cost expressions

Parameter values
Operations research
Computer algebra

Tile sizes
Bound as a function of $S$

```c
for(i = 0; i < Ni; i++)
    for(j = 0; j < Nj; j++)
        for(k = 0; k < Nk; k++)
            C[i][j] += A[i][k] * B[k][j];
```
Upper bound derivation

Input program

Pruning algorithm

Tiling loop permutations

For each permutation

Parametrically tiled program

Polyhedral calculus

Symbolic I/O cost expressions

{(i, j, k), (i, k, j), (k, j, i)}

Tile sizes

Bound as a function of $S$

for(i = 0; i < Ni; i++)
    for(j = 0; j < Nj; j++)
        for(k = 0; k < Nk; k++)
            C[i][j] += A[i][k] * B[k][j];
Upper bound derivation

Input program

Pruning algorithm

Tiling loop permutations

For each permutation

Parametrically tiled program

Polyhedral calculus

Symbolic I/O cost expressions

Parameter values

Operations research

Computer algebra

Tile sizes

Bound as a function of $S$

```
for(i = 0; i < Ni; i++)
for(j = 0; j < Nj; j++)
for(k = 0; k < Nk; k++)
    C[i][j] += A[i][k] * B[k][j];
```

```
for(i1 = 0; i1 < Ni; i1 += Ti)
for(j1 = 0; j1 < Nj; j1 += Tj)
for(k = 0; k < Nk; k++)
    for(i = i1; i < i1 + Ti; i++)
        for(j = j1; j < j1 + Tj; j++)
            C[i][j] += A[i][k] * B[k][j];
```

IO = $N_iN_jN_k (1/T_i + 1/T_j + 1/N_k) T_i T_j + T_i + T_j \leq S$

UB = $N_i N_j (2N_k \sqrt{S} + 1 - 1 + 1)$
Upper bound derivation

Input program

Pruning algorithm

Tiling loop permutations

For each permutation

Parametrically tiled program

Polyhedral calculus

Symbolic I/O cost expressions

Parameter values
- Operations research
- Computer algebra

Tile sizes

Bound as a function of $S$

\[
\text{for } (i = 0; i < N_i; i++) \\
\text{for } (j = 0; j < N_j; j++) \\
\text{for } (k = 0; k < N_k; k++) \\
\quad C[i][j] += A[i][k] * B[k][j];
\]

\[
\{(i, j, k), (i, k, j), (k, j, i)\}
\]

\[
\text{for } (i_1 = 0; i_1 < N_i; i_1 += T_i) \\
\text{for } (j_1 = 0; j_1 < N_j; j_1 += T_j) \\
\text{for } (k = 0; k < N_k; k++) \\
\quad \text{for } (i = i_1; i < i_1 + T_i; i++) \\
\quad \text{for } (j = j_1; j < j_1 + T_j; j++) \\
\quad \quad C[i][j] += A[i][k] * B[k][j];
\]

\[
IO = N_i N_j N_k \left( \frac{1}{T_i} + \frac{1}{T_j} + \frac{1}{N_k} \right)
\]

\[
T_i T_j + T_i + T_j \leq S
\]
Upper bound derivation

Input program

Pruning algorithm

Tiling loop permutations

For each permutation

Parametrically tiled program

Polyhedral calculus

Symbolic I/O cost expressions

Parameter values
Operations research

Tile sizes

Bound as a function of $S$

$\textbf{for}(i = 0; i < N_i; i++)$
$\textbf{for}(j = 0; j < N_j; j++)$
$\textbf{for}(k = 0; k < N_k; k++)$
$C[i][j] += A[i][k] * B[k][j];$

$\{(i, j, k), (i, k, j), (k, j, i)\}$

$\textbf{for}(i1 = 0; i1 < N_i; i1+= Ti)$
$\textbf{for}(j1 = 0; j1 < N_j; j1+= Tj)$
$\textbf{for}(k = 0; k < N_k; k++)$
$\textbf{for}(i = i1; i < i1 + Ti; i ++)$
$\textbf{for}(j = j1; j < j1 + Tj; j ++)$
$C[i][j] += A[i][k] * B[k][j];$

$I O = N_i N_j N_k \left( \frac{1}{T_i} + \frac{1}{T_j} + \frac{1}{N_k} \right)$

$T_i T_j + T_i + T_j \leq S$

$UB = N_i N_j \left( \frac{2N_k}{\sqrt{S+1}} + 1 \right)$
In the paper: Analytical results on several convolutions (Yolo9000) and tensor contractions (TCCG), with matching lower and upper bounds.
Thank you!