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 showing CPU, Fast memory with capacity $S$, and Slow memory with unbounded capacity.]
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

*Diagram with CPU, Fast memory with capacity $S$, and Slow memory with 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

![Diagram of CPU and memory systems](image-url)
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

IOLB (PLDI ’20)
Automated lower bound computation
Lower and Upper Bounds

IOLB (PLDI ’20)
Automated lower bound computation

IOOpt (This paper)

- Improvement of the lower bound algorithm
- Automated upper bound derivation (IOUB)
I/O Upper Bounds

I/O complexity upper bound ⇔ Cost of a particular valid schedule

Untiled matrix multiplication

```c
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
```

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

Tiled matrix multiplication

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

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

→ 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

Tile sizes

Bound as a function of $S$

Computer algebra
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 $S$

$\text{for}(i = 0; i < \text{Ni}; i++)$
$\text{for}(j = 0; j < \text{Nj}; j++)$
$\text{for}(k = 0; k < \text{Nk}; k++)$
$C[i][j] += A[i][k] * B[k][j];$
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 \( S \)

\[
\begin{align*}
\text{for}(i = 0; i < Ni; i++) \\
\text{for}(j = 0; j < Nj; j++) \\
\text{for}(k = 0; k < Nk; k++)
C[i][j] &= A[i][k] \times B[k][j]; \\
\{ (i, j, k), (i, k, j), (k, j, i) \}
\end{align*}
\]
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 \( S \)

```
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];
```

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

```
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];
```
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**

Computer algebra

**Bound as a function of S**

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

\[
T_i T_j + T_i + T_j \leq S
\]
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 $S$

\begin{align*}
\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];
\end{align*}

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

\begin{align*}
\text{for} (i1 = 0; i1 < N_i; i1+=Ti) \\
\text{for} (j1 = 0; j1 < N_j; j1+=Tj) \\
\text{for} (k = 0; k < N_k; k++) \\
\quad \text{for} (i = i1; i < i1+Ti; i +=) \\
\quad \quad \text{for} (j = j1; j < j1+Tj; j +=) \\
\quad \quad \quad C[i][j] += A[i][k] * B[k][j];
\end{align*}

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

\[ T_i T_j + T_i + T_j \leq S \]

\[ UB = N_i N_j \left( \frac{2N_k}{\sqrt{S+1} - 1} + 1 \right) \]
Matrix multiplication I/O complexity

\[
\frac{2N_i N_j (N_k - 1)}{\sqrt{S}} \leq IO_{mm} \leq \frac{2N_i N_j N_k}{\sqrt{S+1} - 1}
\]

In the paper: Analytical results on several convolutions (Yolo9000) and tensor contractions (TCCG), with matching lower and upper bounds.
Multi-level tiling driven by IOOpt model

Microkernel: highly tuned “basic block” (vectorization, register reuse, instruction-level parallelism)

I/O: model-driven tiling (IOOpt)

```c
for (kt = 0; kt < Nk; kt+=128)
  for (it = 0; it < Ni; it+=32)
    for (jt = 0; jt < Nj; jt+=6)
      for (k = kt; k < kt+128; k++)
        \mu\text{kernel\_gemm}(A, B, C, i1, j1, k)
```

CPU: microkernel selection
Performance comparison between AutoTVM, oneDNN, and TTile+TVM for AVX512 (Intel Xeon Gold 6130), shown as percentage of machine peak. 32 threads were used, no hyperthreading
Thank you!