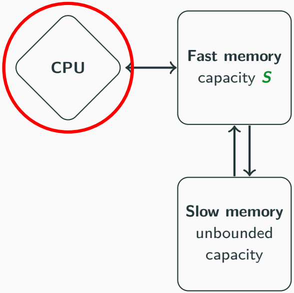
IOOpt: Automatic Derivation of I/O Complexity Bounds for Affine Programs

Auguste Olivry Guillaume Iooss Nicolas Tollenaere
Atanas Rountev P. Sadayappan Fabrice Rastello

June 2021

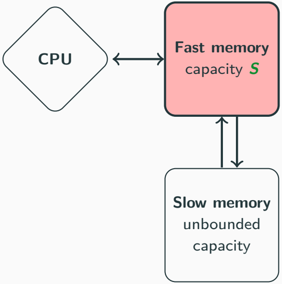
What is I/O complexity?

- Arithmetic complexity $=$ $\#$ of operations



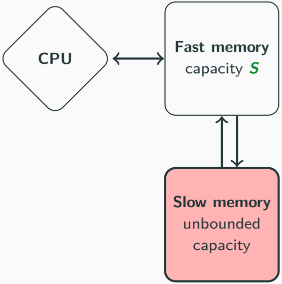
What is I/O complexity?

- Arithmetic complexity = # of operations



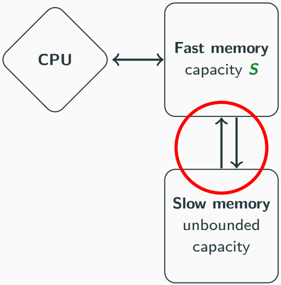
What is I/O complexity?

- Arithmetic complexity = # of operations



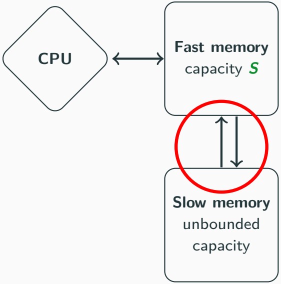
What is I/O complexity?

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

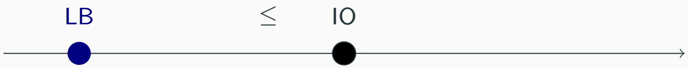


What is I/O complexity?

- 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



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

Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs*

Auguste Olivry
Univ. Grenoble Alpes,
CNRS, Inria, Grenoble INP, LIG
38000 Grenoble, France

Julien Langou
University of Colorado Denver
Denver, CO, USA

Louis-Noël Pouchet
Colorado State University
Fort Collins, CO, USA

P. Sadayappan
University of Utah
Salt Lake City, UT, USA

Fabrice Rastello
Univ. Grenoble Alpes,
Inria, CNRS, Grenoble INP, LIG
38000 Grenoble, France

Abstract

Researchers and practitioners have for long worked on improving the computational complexity of algorithms, focusing on reducing the number of operations needed to perform a computation. However the hardware trend nowadays clearly shows a higher performance and energy cost for data movements than computations: quality algorithms have to minimize data movements as much as possible.

The theoretical operational complexity of an algorithm is a function of the total number of operations that must be executed, regardless of the order in which they will actually be executed. But theoretical data movement (or, I/O) complexity is fundamentally different: one must consider all possible legal schedules of the operations to determine the minimal number of data movements achievable, a major theoretical challenge. I/O complexity has been studied via complex manual proofs, *e.g.*, refined from $\Omega(n^3/\sqrt{S})$ for matrix-multiply on a cache size S by Hong & Kung to $2n^3/\sqrt{S}$ by Smith et al. While asymptotic complexity may be sufficient to compare I/O potential between broadly different algorithms, the accuracy of the reasoning depends on the tightness of these I/O lower bounds. Precisely, exposing constants is essential to enable precise comparison between different algorithms: for example the $2n^3/\sqrt{S}$ lower bound allows to demonstrate the optimality of panel-panel tiling for matrix-multiplication.

We present the first static analysis to automatically derive non-asymptotic parametric expressions of data movement lower bounds with scaling constants, for arbitrary affine computations. Our approach is fully automatic, assisting algorithm designers to reason about I/O complexity and make educated decisions about algorithmic alternatives.

CCS Concepts: **• Theory of computation → Design and analysis of algorithms**; **• Software and its engineering → Automated static analysis**.

Keywords: Data access complexity; I/O lower bounds; Static analysis; Affine programs

1 Introduction

The performance impact of operations and data movement latencies in current architectures can often be effectively masked by using hardware-pipelined implementations. But the volume of data movements required by even an idealized implementation of an algorithm will impose fundamental limits: any implementation of that algorithm will have its performance and energy requirements bounded by this limit [12, 18, 25, 26, 28, 29]. Providing algorithm designers with tools to characterize this fundamental limit is crucial.

Memory movements can be efficiently tracked for a particular algorithm *implementation*, and it is standard practice for performance debugging [1]: Hardware counters can be used to measure cache misses and data traffic. But two different implementations of the same algorithm may have dramatically different memory movement profiles: for example a carefully tiled implementation of matrix multiplication would significantly reduce cache misses versus a naive, untiled one.

In general, determining whether an implementation is sub-optimal or whether the fundamental nature of the algorithm is the limiting factor for the observed cache miss count is crucial. We propose an *automatic* system to answer this question, potentially alleviating the need for the algorithm designers to produce a concrete optimized implementation. As we specifically target the production of non-asymptotic I/O lower bounds, our system also makes it possible for performance experts to reason about the optimality of their implementations with respect to data movement.

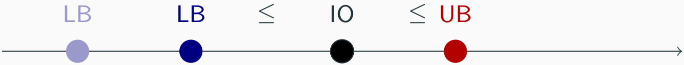
Our fully implemented framework IOLB (for *I/O Lower Bounds*) automatically derives parametric lower bounds with

*This work was supported in part by the U.S. National Science Foundation awards 1645514, 1645599, 1750399 and 1816793.

PLDI '20, June 15–20, 2020, London, UK
2020. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '20), June 15–20, 2020, London, UK*, https://doi.org/10.1145/3385412.3385989.

1

Lower and Upper Bounds



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


Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs*

Auguste Olivry, Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
Julien Langou, University of Colorado Denver, Denver, CO, USA
Louis-Noël Pouchet, Colorado State University, Fort Collins, CO, USA
P. Sadayappan, University of Utah, Salt Lake City, UT, USA
Fabrice Rastello, Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Abstract

Researchers and practitioners have for long worked on improving the computational complexity of algorithms, focusing on reducing the number of operations needed to perform a computation. However the hardware trend nowadays clearly shows a higher performance and energy cost for data movements than computations: quality algorithms have to minimize data movements as much as possible.

The theoretical operational complexity of an algorithm is a function of the total number of operations that must be executed, ...

... designers to reason about I/O complexity and make educated decisions about algorithmic alternatives.

CCS Concepts: **• Theory of computation → Design and analysis of algorithms; • Software and its engineering → Automated static analysis.**

Keywords: Data access complexity; I/O lower bounds; Static analysis; Affine programs

1 Introduction

IOOpt (This paper)

- Improvement of the lower bound algorithm
- Automated upper bound derivation (IOUB)

I/O Upper Bounds

I/O complexity upper bound $\Leftrightarrow$ Cost of a particular valid schedule

Untiled matrix multiplication

```
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: $O(N^3)$

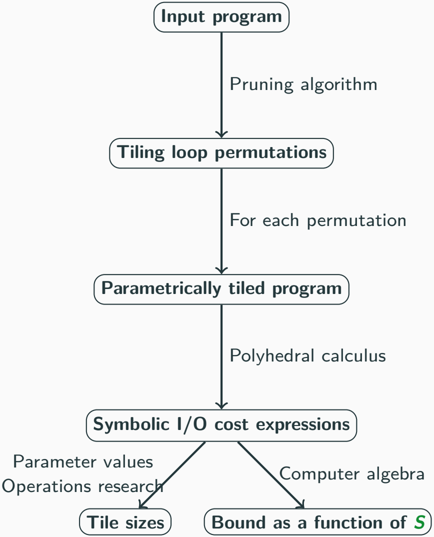
Tiled matrix multiplication

```
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];
```

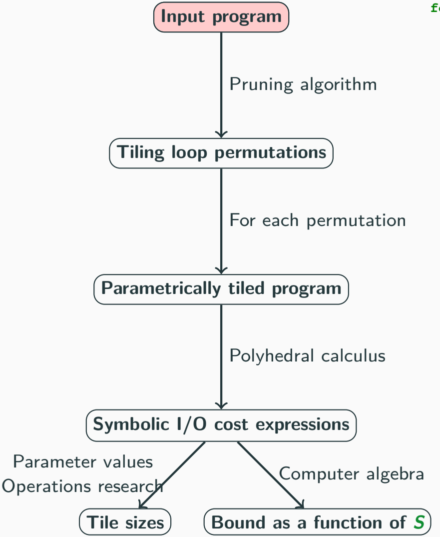
I/O cost: $O(\frac{N^3}{\sqrt{S}})$

$\rightarrow$ How to automatically compute I/O cost for a given schedule?

Upper bound derivation

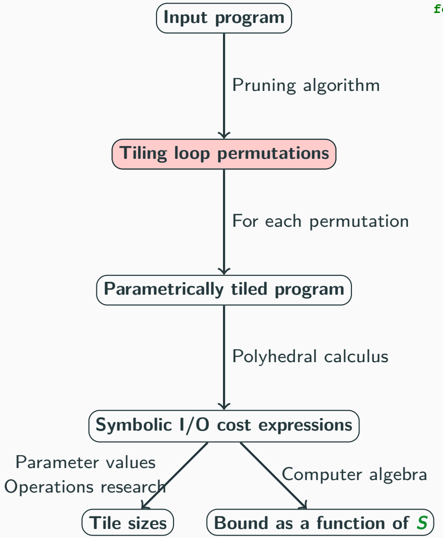


Upper bound derivation




```
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];
```

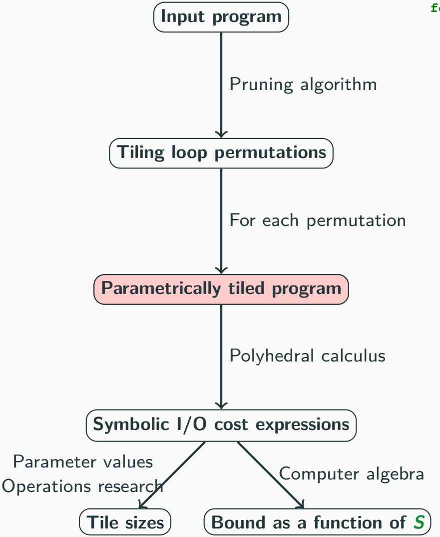
Upper bound derivation




```
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];
```

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

Upper bound derivation




```
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

```
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];
```

Pruning algorithm

Tiling loop permutations

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

For each permutation

Parametrically tiled program

```
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];
```

Polyhedral calculus

Symbolic I/O cost expressions

$$IO = N_i N_j N_k \left(\frac{1}{T_i} + \frac{1}{T_j} + \frac{1}{N_k} \right)$$
$$T_i T_j + T_i + T_j \leq S$$

Parameter values
Operations research

Tile sizes

Computer algebra

Bound as a function of *S*

Upper bound derivation

Input program

```
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];
```

Pruning algorithm

Tiling loop permutations

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

For each permutation

Parametrically tiled program

```
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];
```

Polyhedral calculus

Symbolic I/O cost expressions

$$IO = N_i N_j N_k \left(\frac{1}{T_i} + \frac{1}{T_j} + \frac{1}{N_k} \right)$$
$$T_i T_j + T_i + T_j \leq S$$

Parameter values
Operations research

Tile sizes

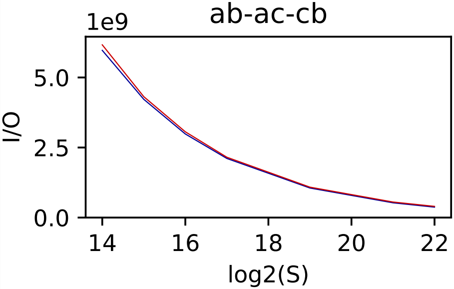
Computer algebra

Bound as a function of *S*

$$UB = N_i N_j \left(\frac{2N_k}{\sqrt{S+1}-1} + 1 \right)$$

Matrix multiplication I/O complexity

$$\frac{2N_iN_j(N_k-1)}{\sqrt{S}} \leq IO_{mm} \leq \frac{2N_iN_jN_k}{\sqrt{S+1}-1}$$



In the paper: Analytical results on several convolutions (Yolo9000) and tensor contractions (TCCG), with matching lower and upper bounds

TTile: Highly Optimized Tensor Computations

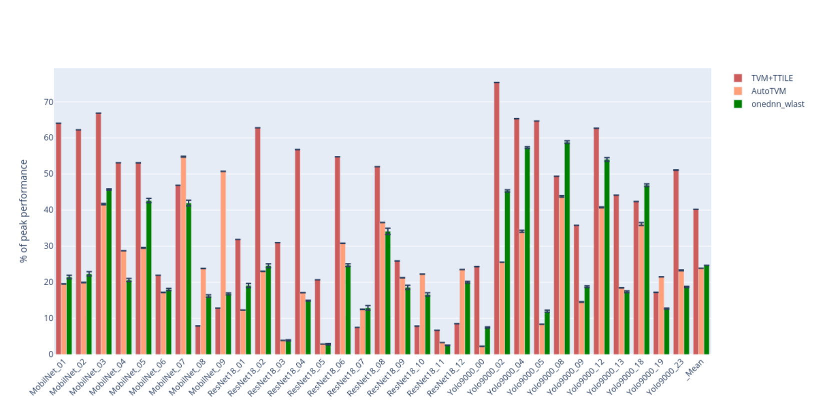
- Multi-level tiling driven by IOOpt model
- Microkernel: highly tuned "basic block" (vectorization, register reuse, instruction-level parallelism)

I/O: model-driven tiling (IOOpt)

```
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++)
        μkernel_gemm(A, B, C, i1, j1, k)
```

CPU: microkernel selection

TTile: Highly Optimized Tensor Contractions



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

Demo

Thank you!