Improving LLVM Instrumentation Overheads

LLVM-Performance @ CGO 2017

Dr. Brian P. Railing

Overview of Talk

- **Contech's Task Graph representation**
- General instrumentation approach for Contech
- Overhead reduction techniques

Objectives of Parallel Program Representation

- **A common representation needs**
  - What was executed
  - What was accessed
  - In what order did threads execute

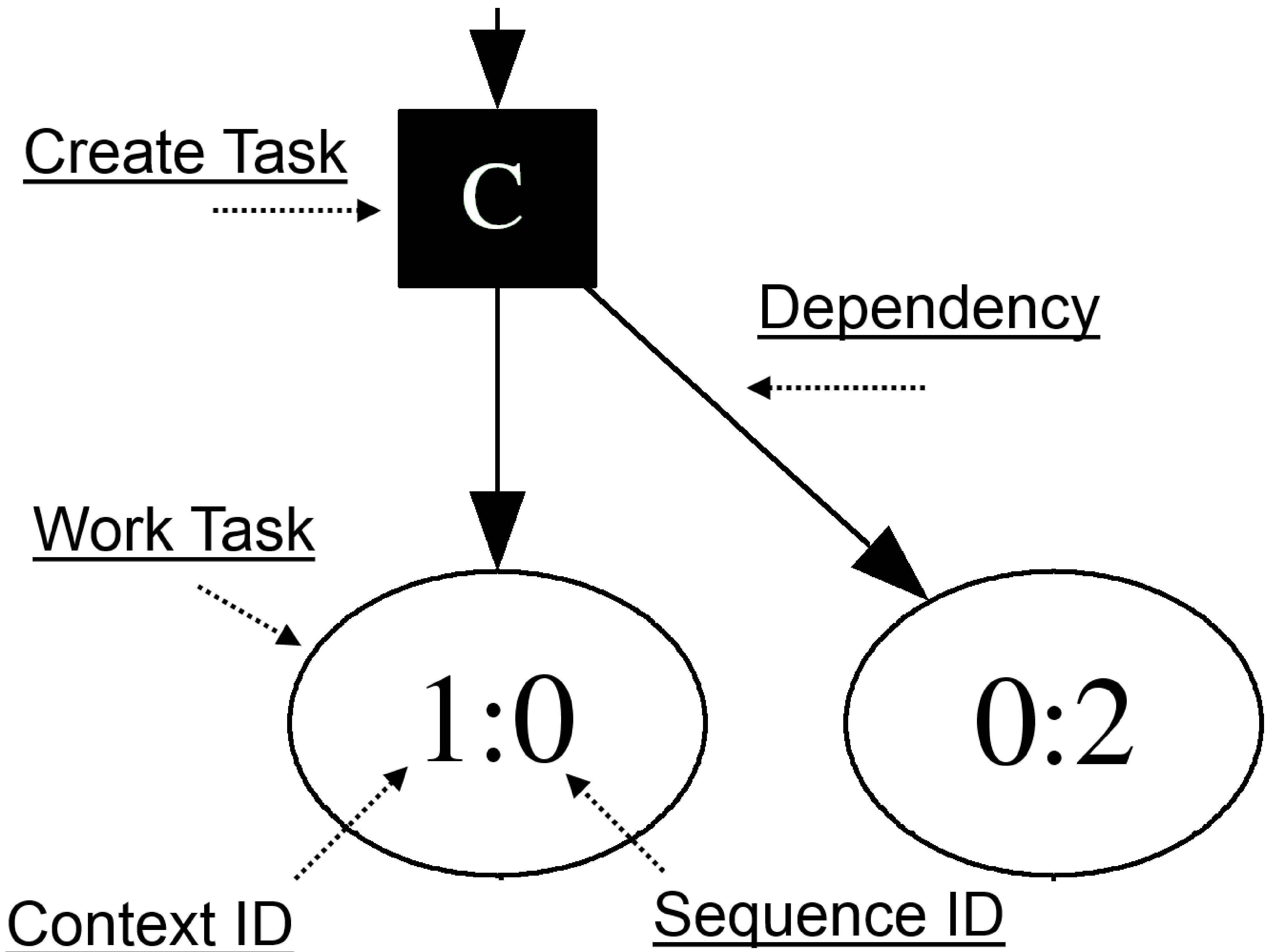
- **Generate the representation with no user intervention**
  - Without constraint of language, library, or structure

- **Without recording architecture / runtime effects**
  - Context switches
  - Consistency model
  - Cache Effects
  - …

Contech's Task Graph Representation

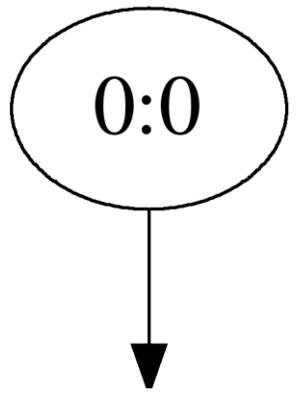
- **Task Graphs are directed, acyclic graphs containing**
  - Nodes partitioned based on type
  - Edges as scheduling dependencies
  - Nodes contain lists of actions and data
  - Other graph annotations such as start / end time

Task Graph Legend



Task Graph Example

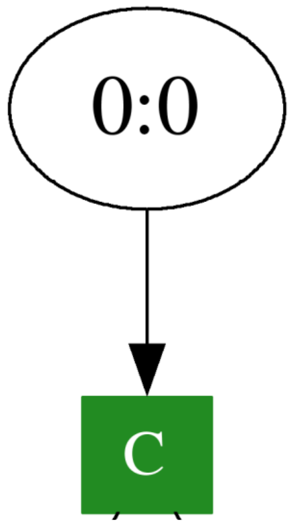
```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}
```




```
fib(2);
```

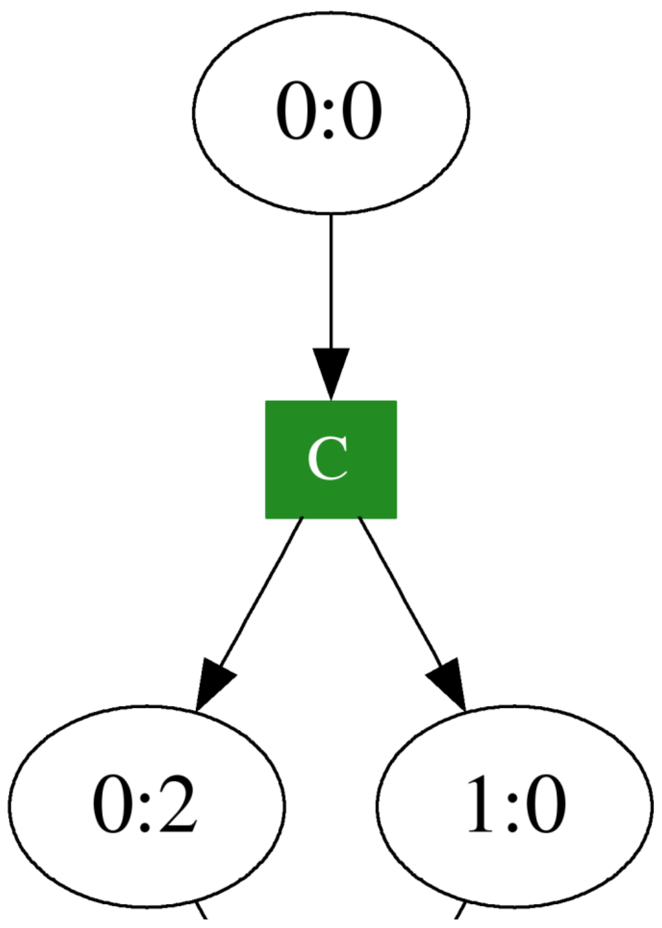
Task Graph Example

```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}

fib(2);
```



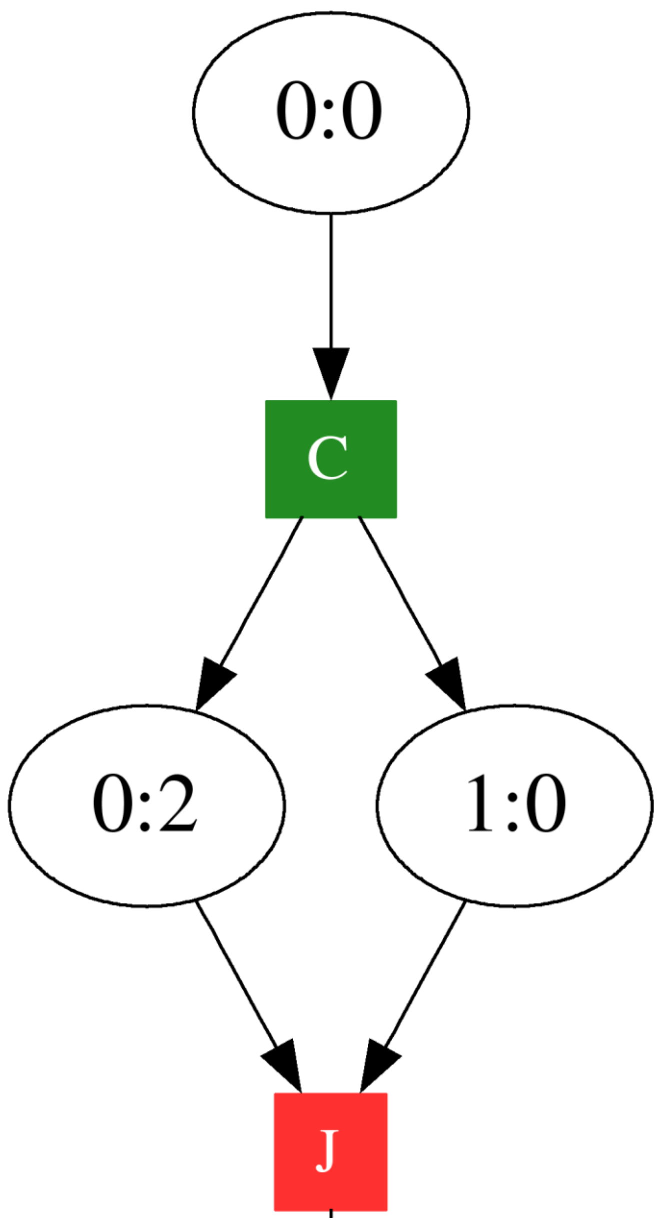
Task Graph Example

```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}

fib(2);
```



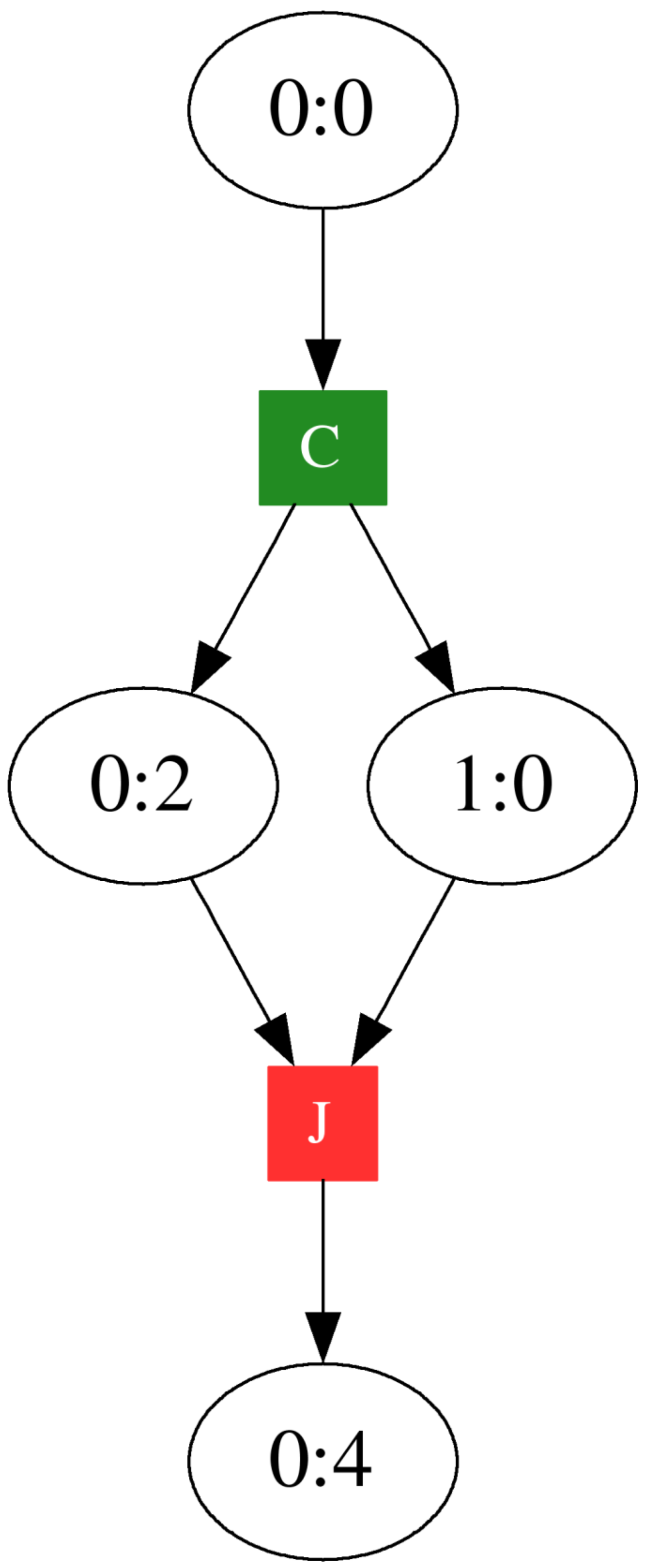
Task Graph Example

```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}

fib(2);
```



Task Graph Example

```
int fib(int n) {
  if (n < 2)
    return n;
  int a = cilk_spawn fib(n-1);
  int b = fib(n-2);
  cilk_sync;
  return a + b;
}

fib(2);
```



Parallel Program Diversity

- **Language Diversity**
  - **C, C++, Fortran**, Java, Go, Rust, X10, …
- **Runtime Diversity**
  - **Pthreads, OpenMP, MPI,** *Cilk*, Galois, Legion, CnC, …
- **Pattern Diversity**
  - **Regular, pipelines,** *graphs*, Map-reduce, Gather-scatter, …
- **Architecture Diversity**
  - **32- / 64-bit x86, ARM**, MIPS, Power, …

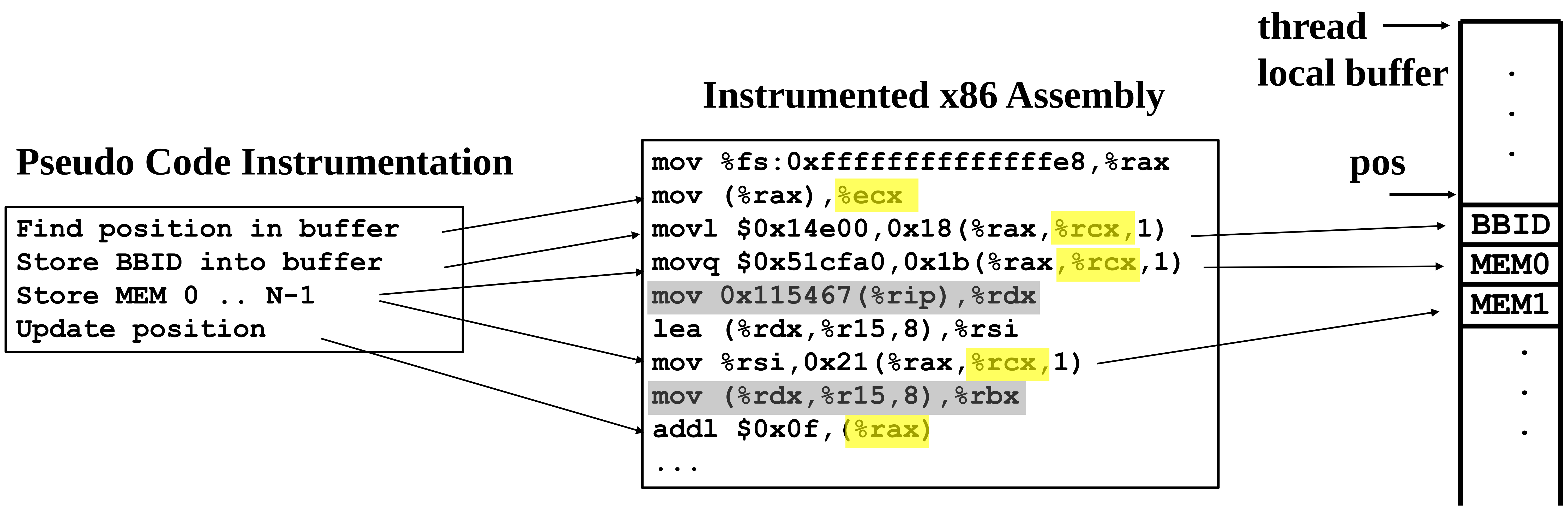
Overview of Talk

- Contech's Task Graph representation
- **General instrumentation approach for Contech**
- Overhead reduction techniques

LLVM Instrumentation Design

- **Compile the source language into LLVM IR**
- **Instrument each basic block**
  - Record its execution (i.e. control flow)
  - Record memory operations
  - Record other operations

Instrumentation Design



Buffer Checking and Queuing

- **Thread local buffer capacity checks**
  - Check for 0.1% space remaining (1KB)
  - Only put checks in some basic blocks
- **Queuing buffer into global list**
  - Global lock and push back buffer
  - Allocate a new buffer (or reuse)
- **Queued buffers will grow to use memory**

Overview of Talk

- Contech's Task Graph representation
- General instrumentation approach for Contech
- **Overhead reduction techniques**

Presentation Thesis

- **Memory traffic from instrumentation dominates overheads**
- **Each instrumented thread generates 100MB/s – 1GB/s**
- **Basic blocks are 90+% of trace**
  - And each basic block event is mostly memory operations

Recording a Memory Operation

- **What can a memory operation trace record**
  - Load/Store
  - Address
  - Size / Type
  - Value

Prior Static Analysis in Contech

- **Basic blocks are consistent in memory operations**
  - If we record basic blocks, then load/store and size/type is unchanged on each execution
  - Record the load/store and size/type once in basic block info table

- **64-bit Addresses are only 6 bytes**

```
00 01 02 03 04 05 00 01 02 03 04 05 00 01 02 03 04 05 06 07
```

Current Static Analysis

- **Not all addresses are required.**
  - Addresses are constant
  - Or are constant offsets from other addresses.

Detecting Similar Addresses

- **For each basic block**
  - For each memory operation
    - Check if any prior operation in this basic block has a similar address calculation

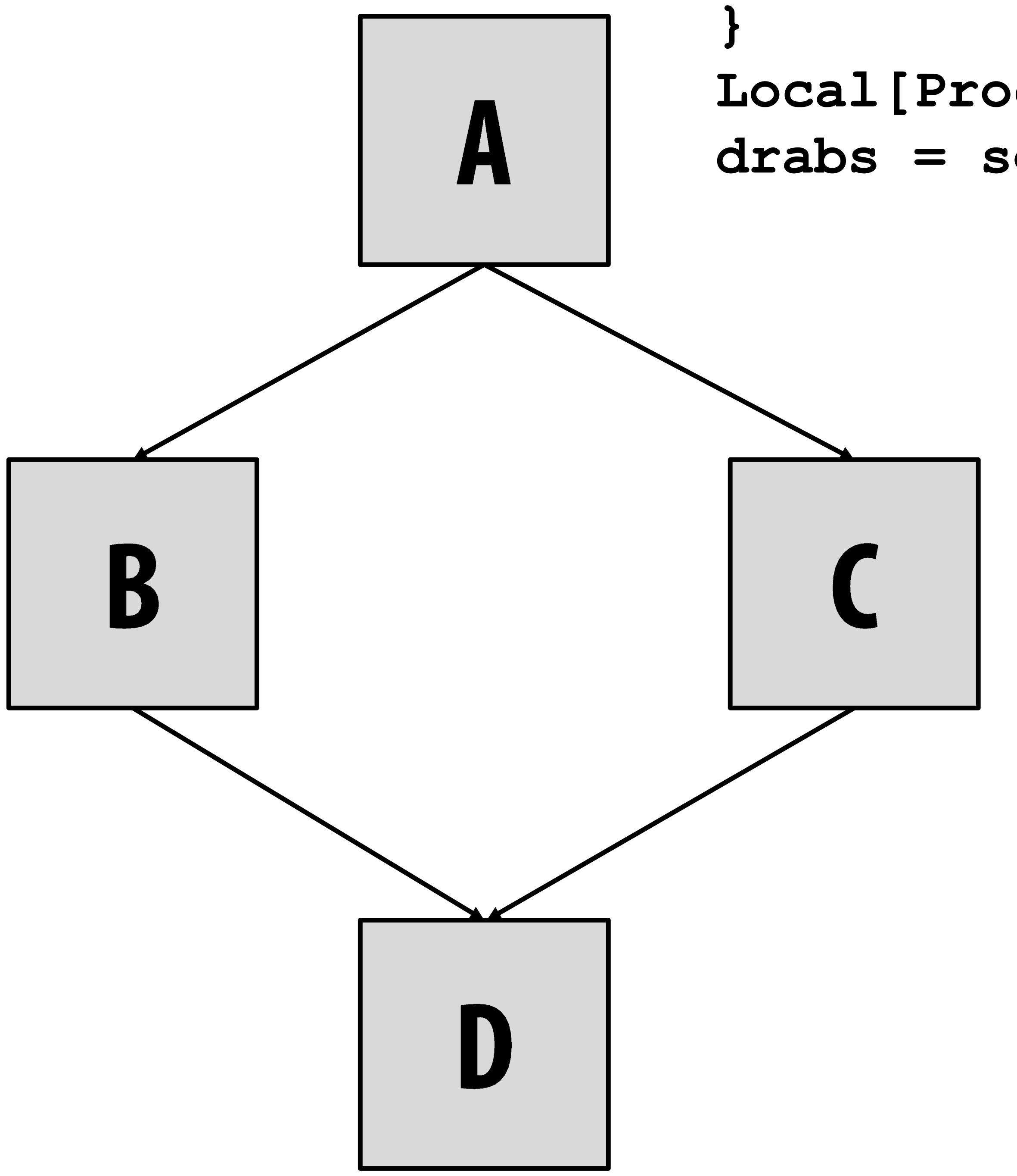
- **Similar Address Calculations**
  - Is it this a getelementptr instruction?
  - Does each component match?
  - If not, is the component a constant value?
    - Accumulate constant differences

- **Store memory operation indices and constant differences into basic block info table**

Similar Address Problem (barnes)

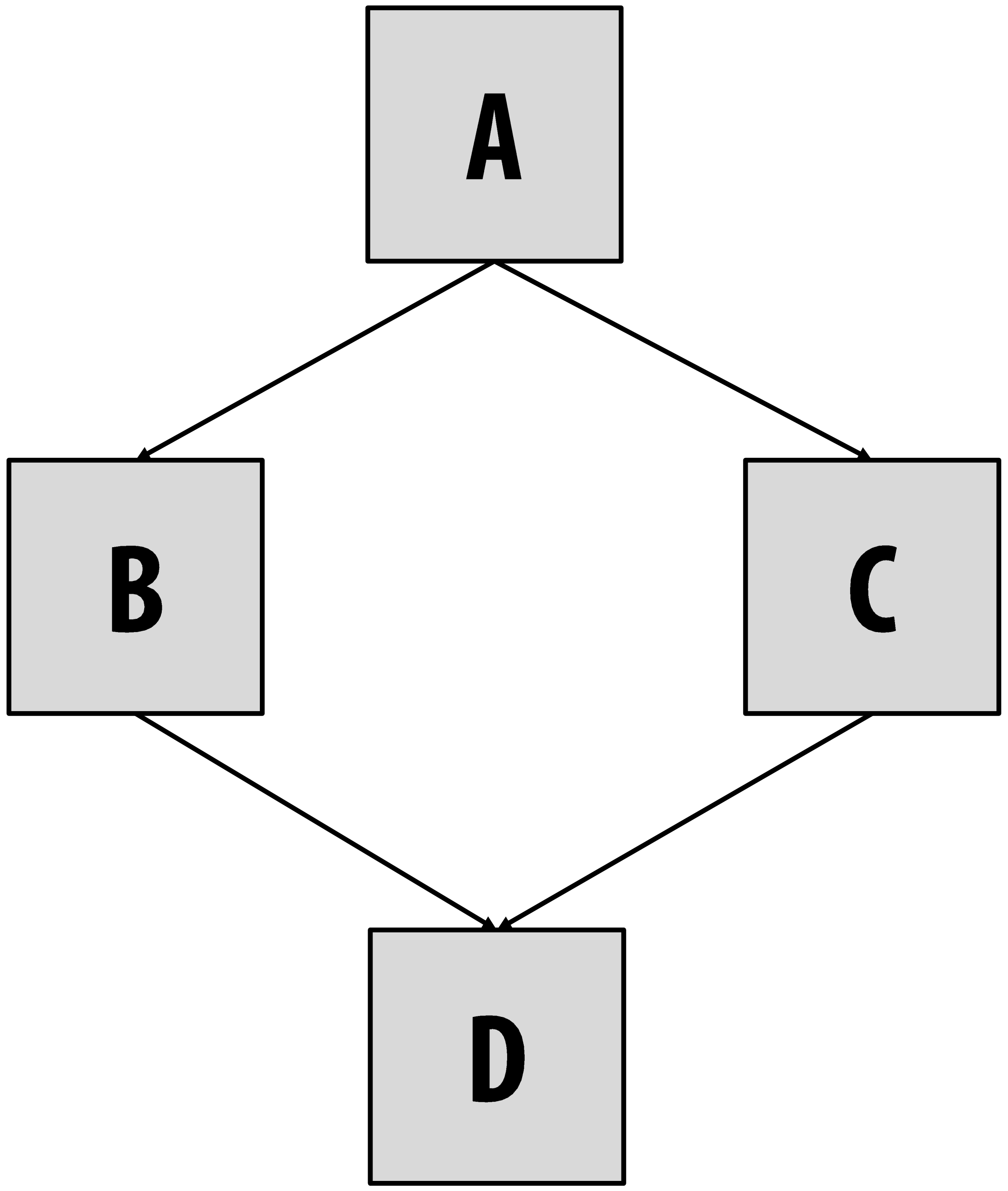
- Conditional code in one path
- Load/Store in tail block

```
if (p != Local[ProcessId].pmem) {
    SUBV(Local[ProcessId].dr,
         Pos(p),
         Local[ProcessId].pos0);
    DOTVP(Local[ProcessId].drsq,
          Local[ProcessId].dr,
          Local[ProcessId].dr);
}
Local[ProcessId].drsq += epssq;
drabs = sqrt((double) Local[ProcessId].drsq);
```



Tail Duplication

- Duplicate the tail block to enlarge the scope for finding similar addresses
- Merge it with each of the predecessor blocks



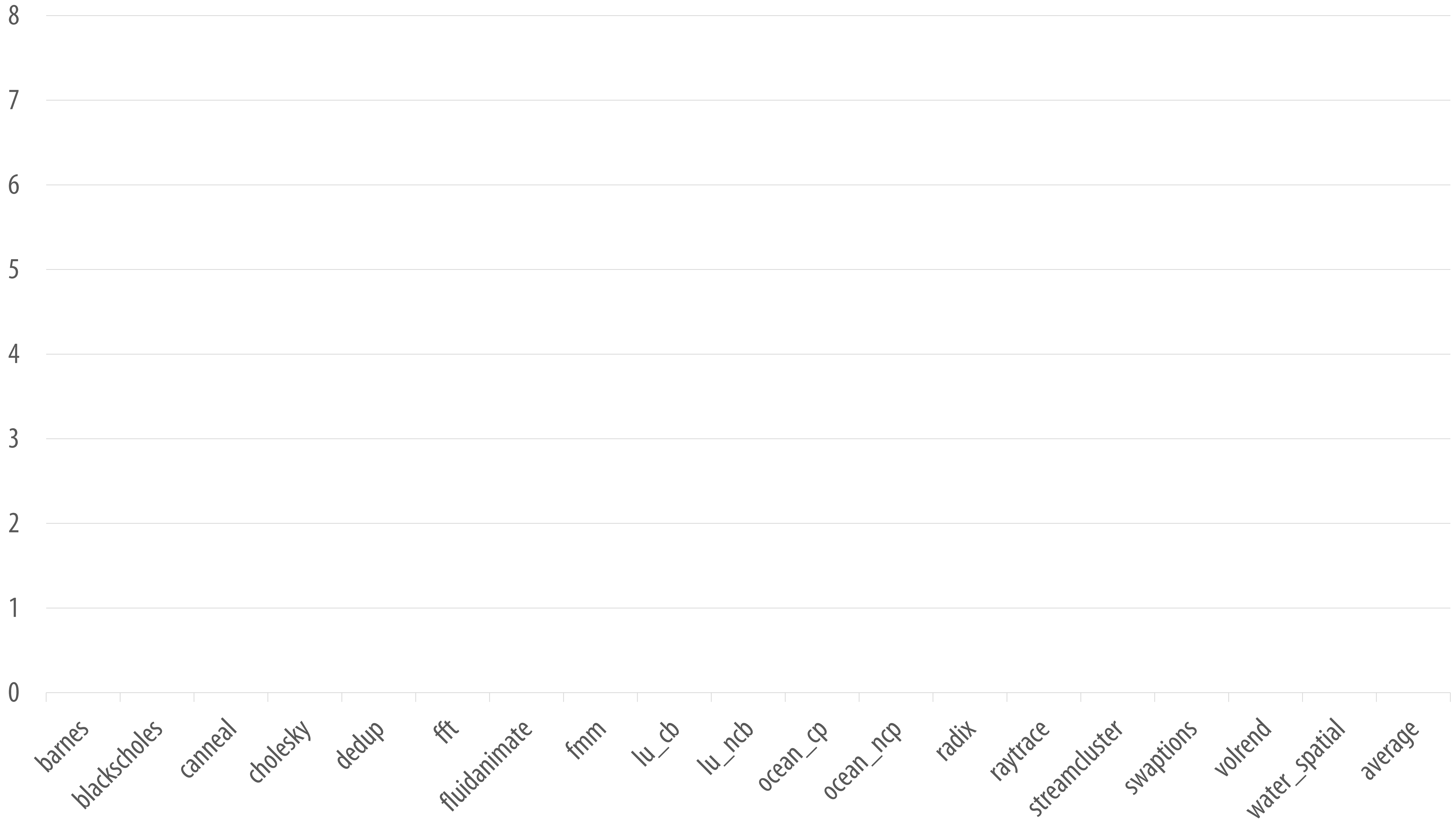
Tail Duplication Algorithm

- **Determine if the tail block is valid for duplication**
  - Not the return block
  - No address taken
  - Etc.

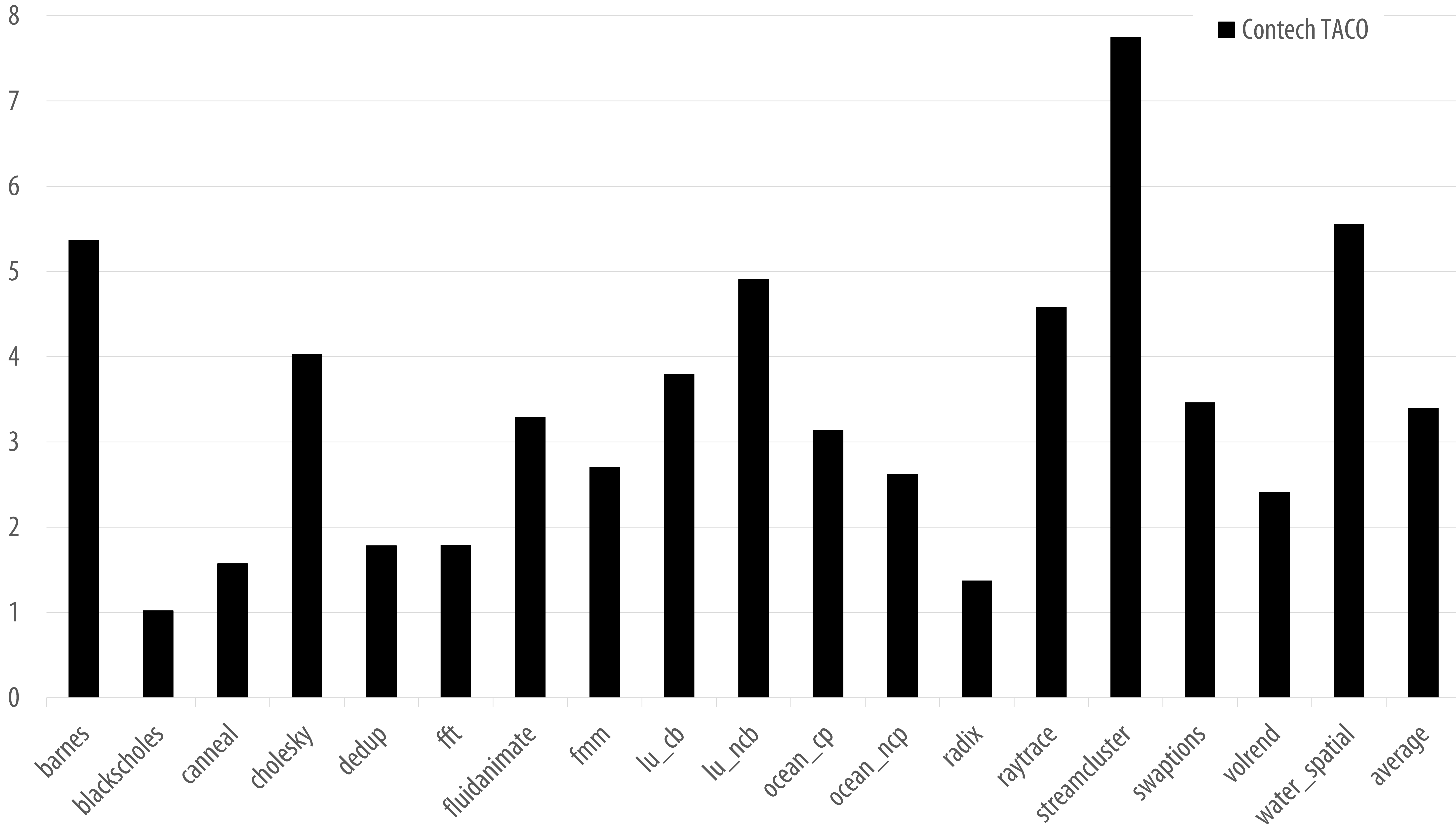
- **Determine that each predecessor is valid**
  - Unconditional branch to tail block

- **Duplicate and Merge**
  - Duplicate the tail block
  - Create / update PHI nodes as appropriate

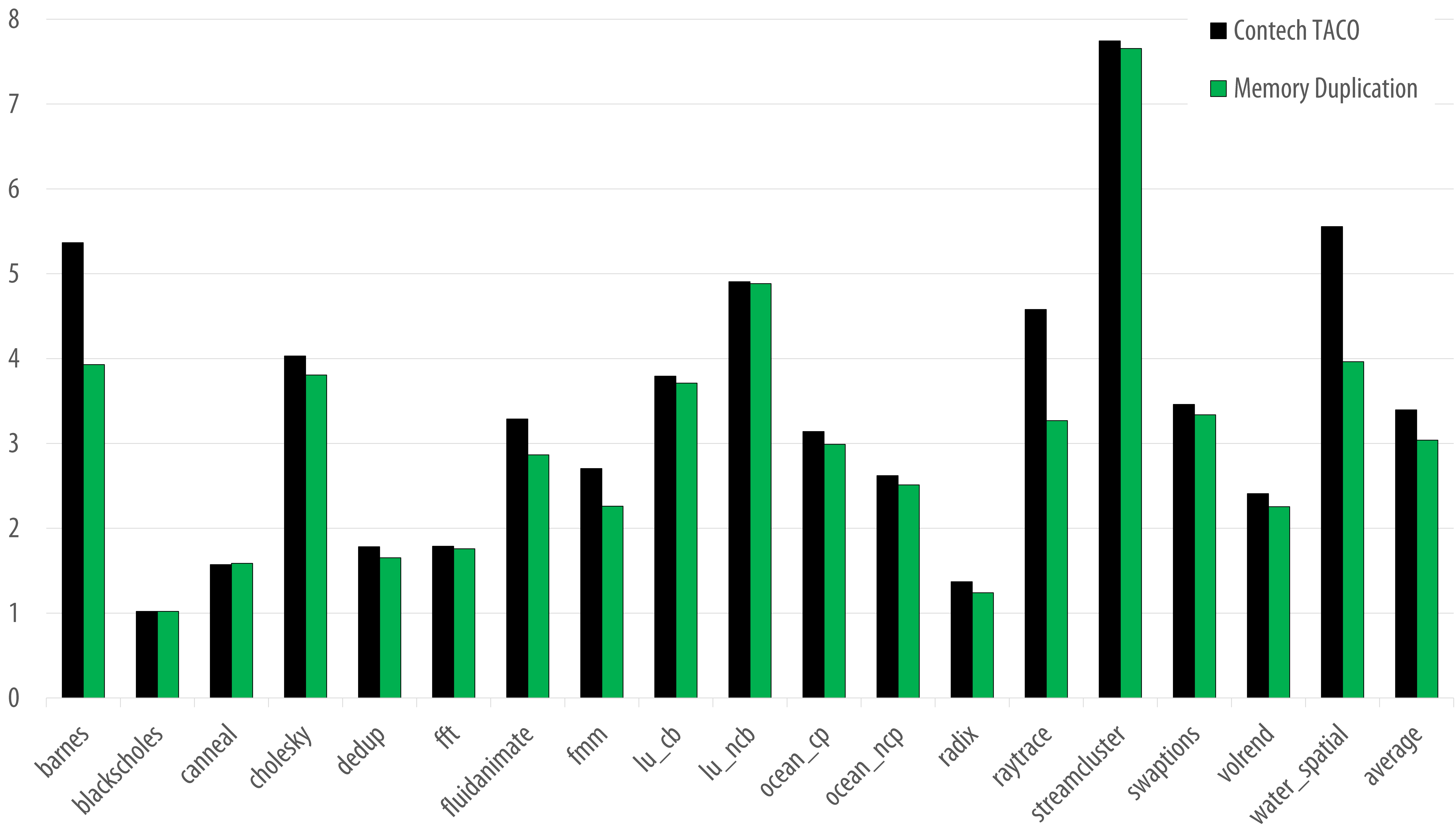
Instrumentation Performance Comparison



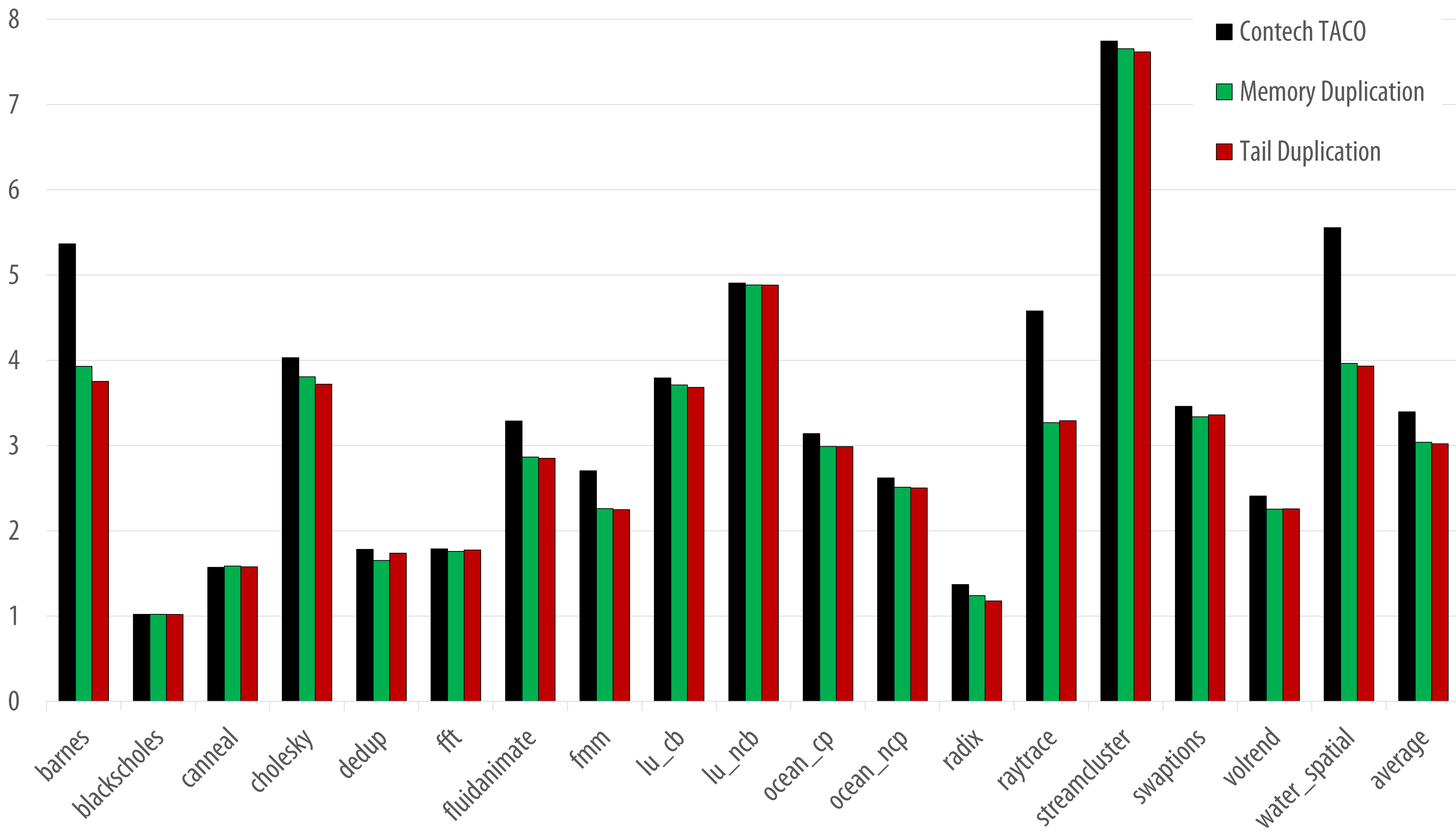
Instrumentation Performance Comparison



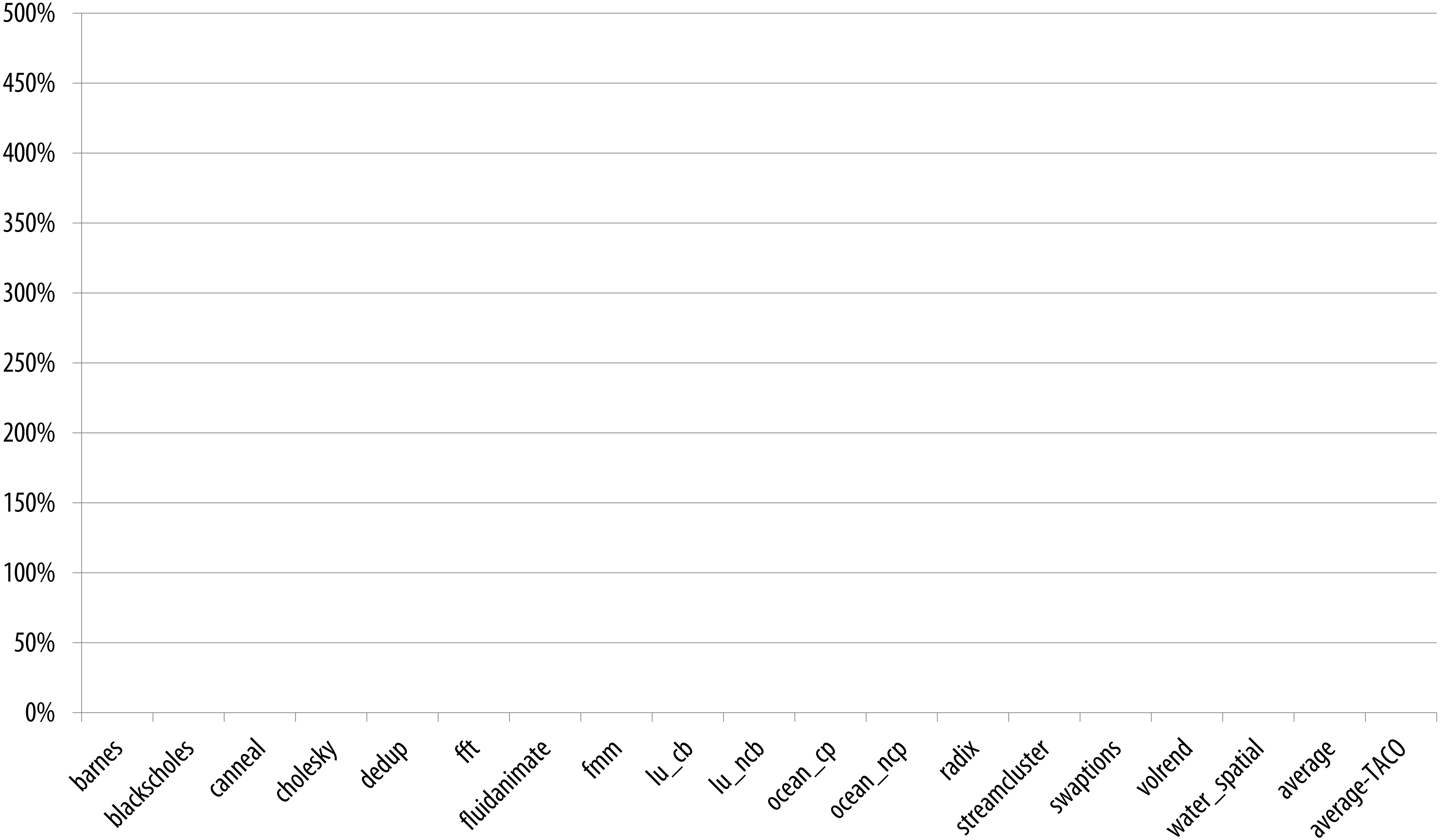
Instrumentation Performance Comparison



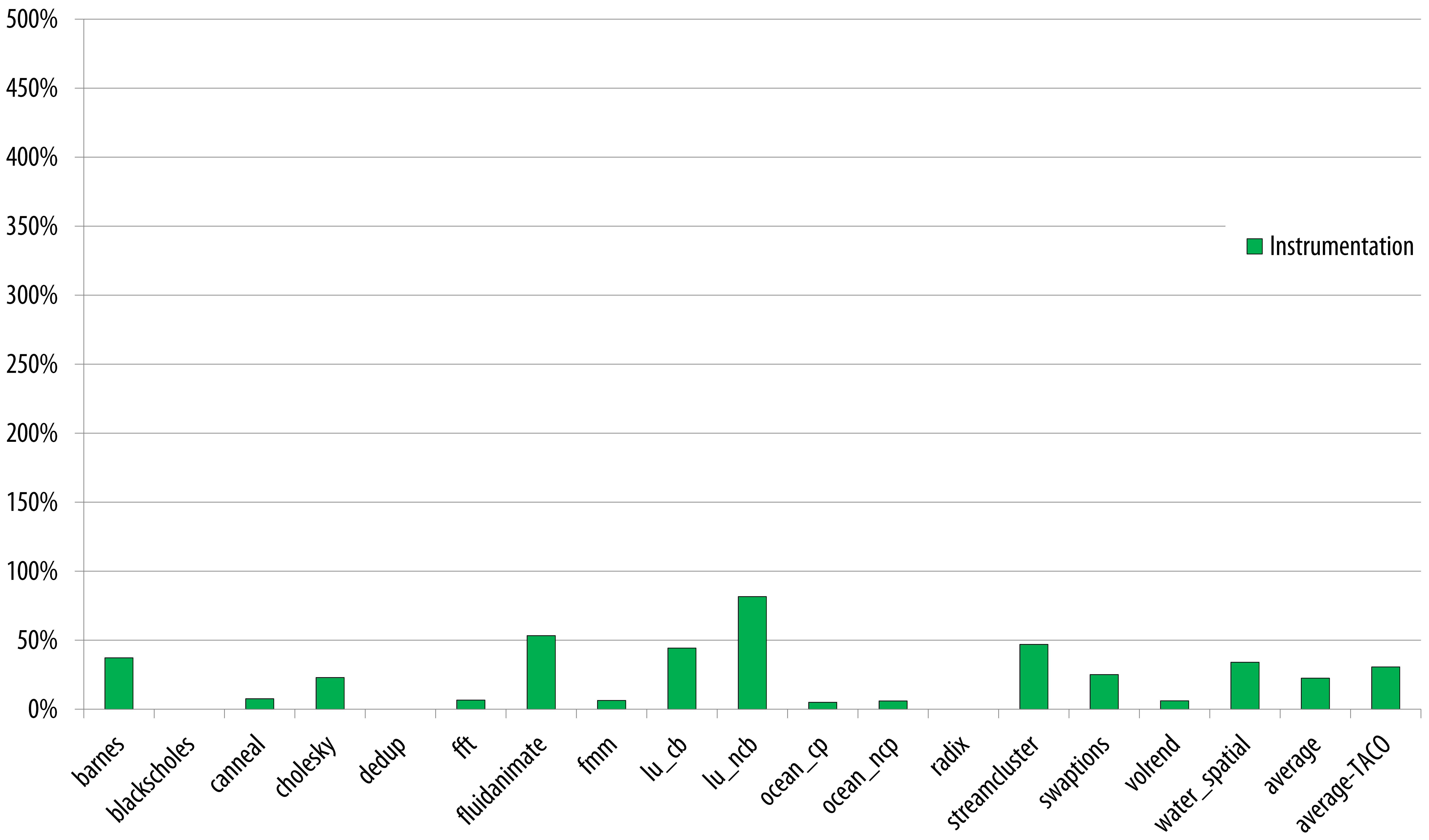
Instrumentation Performance Comparison



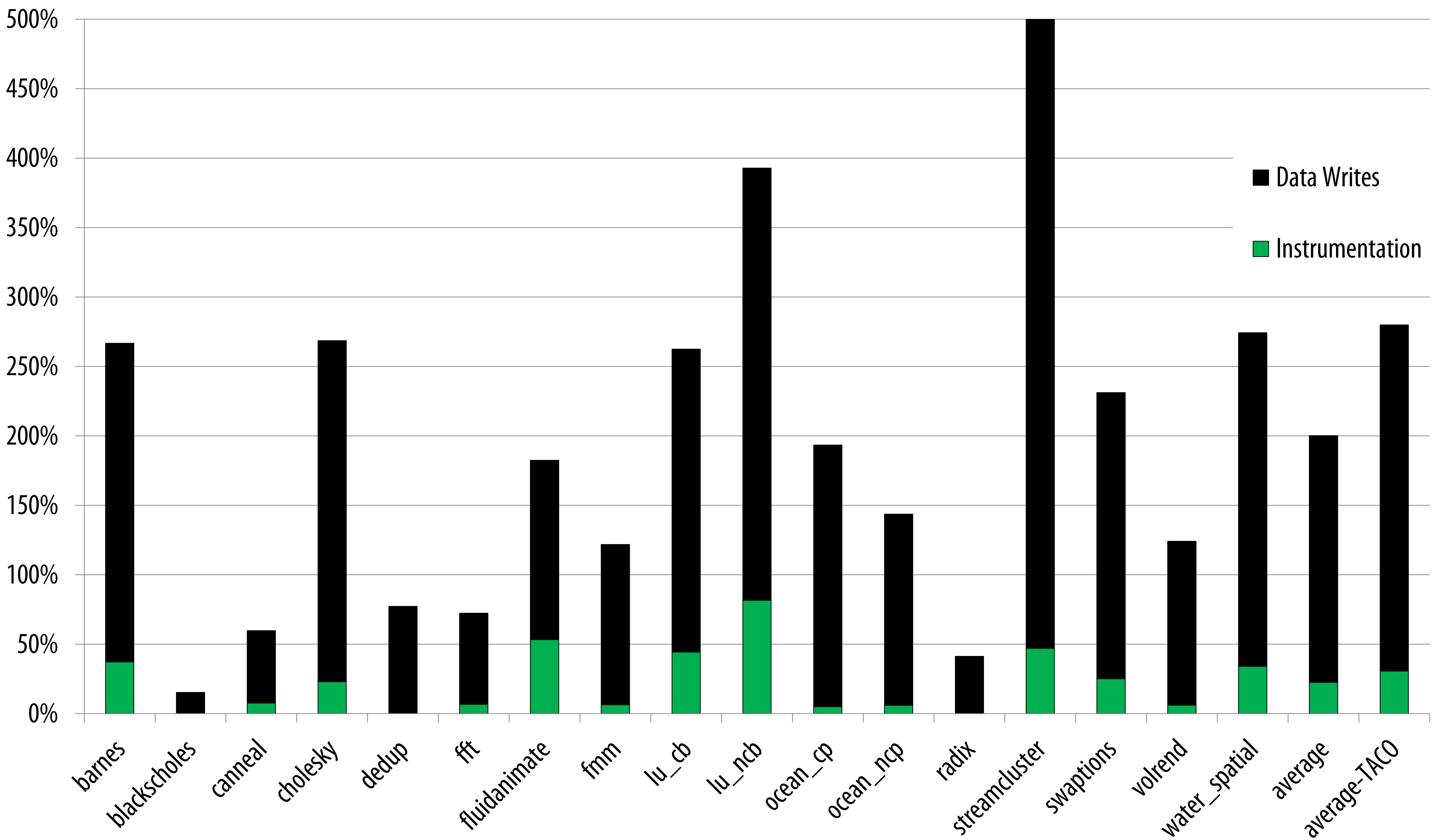
Instrumentation Overhead



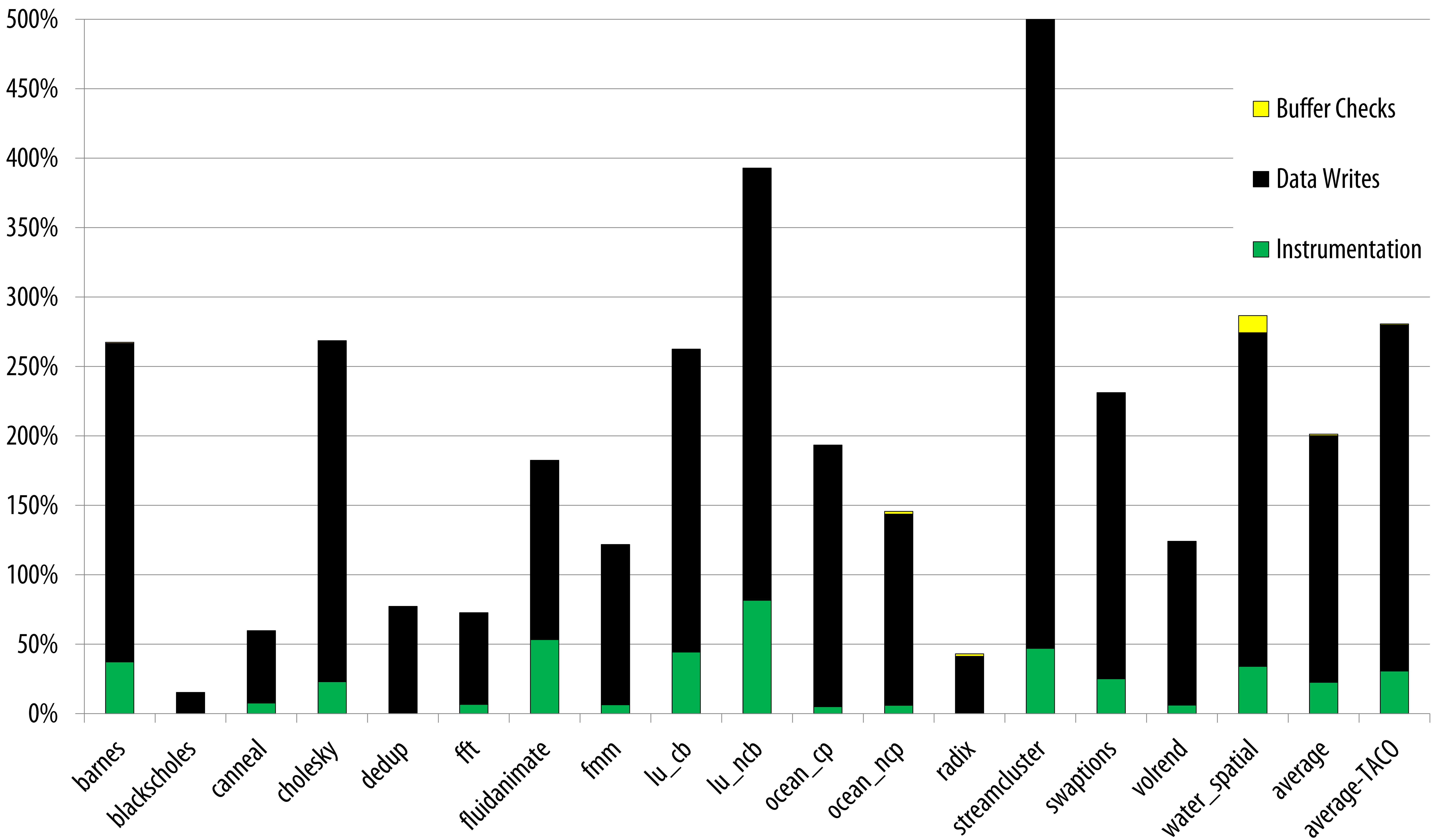
Instrumentation Overhead



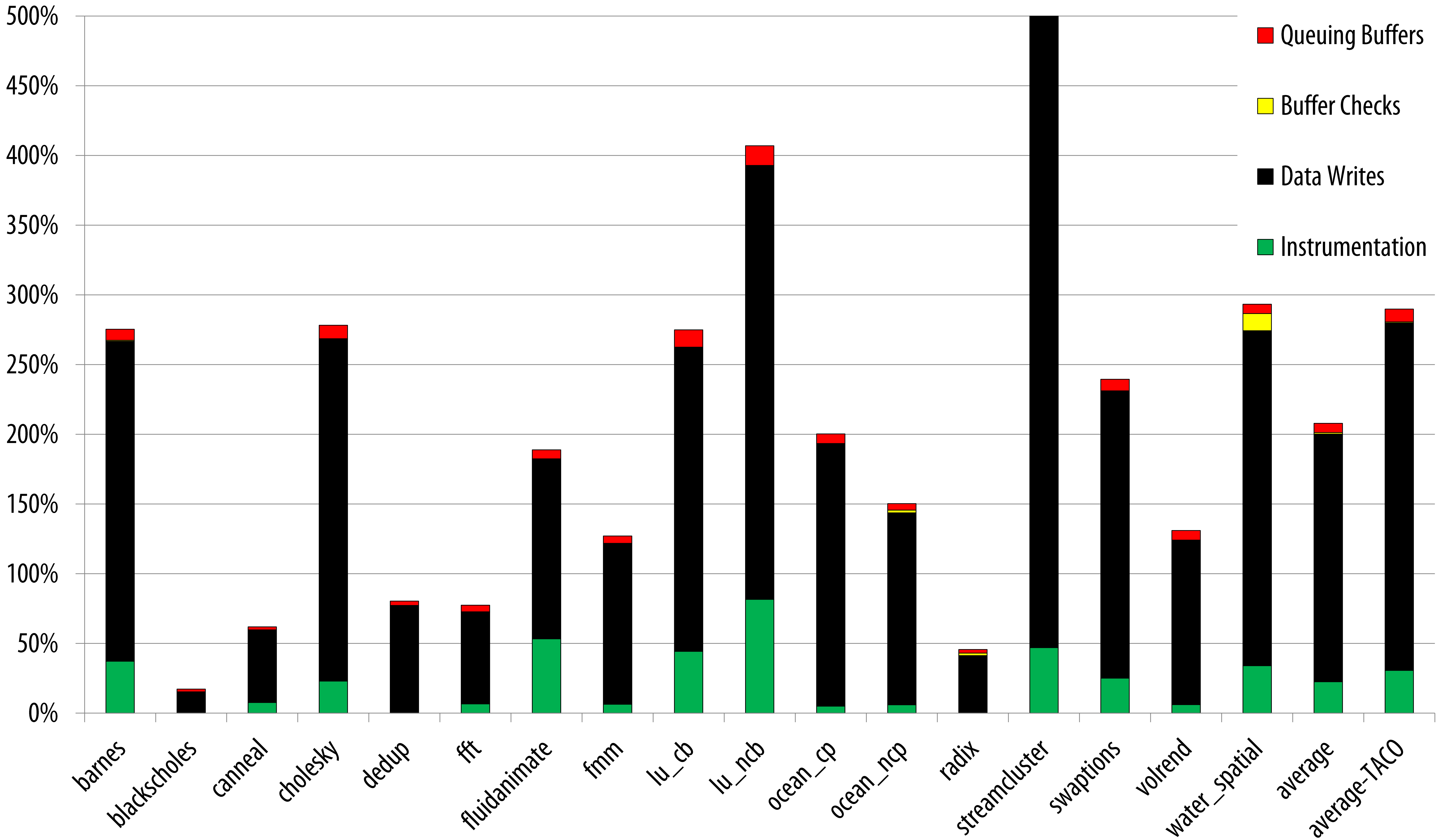
Instrumentation Overhead



Instrumentation Overhead



Instrumentation Overhead



Conclusion

- **Prior work reduced instrumentation instructions required**
- **Prior work minimized instrumented thread interactions**
  - Tickets to order locks and barrier operations
  - Maximize usage of buffers
- **Instrumentation performance is often memory bandwidth constrained**
  - Minimize the size of records
  - Find redundant data and elide
- **LTO is very valuable**

Future Work

- **Global Variables**
  - Address is known at link time, how to record this
- **Memory Operations in a Loop**
  - Base pointer + offset function to reconstruct addresses
- **Release set of collected task graphs**

Code Available

- http://bprail.github.io/contech/

Hardware Configuration

- **Intel Xeon E3-1240v5 (Skylake)**
  - 3.50 GHz Quad-core, 2-way Hyperthreading
- **32 GB Main Memory**
- **256 GB NVMe M.2 PCIe SSD**
  - minimal speedup versus tmpfs or local storage