MANCHESTER 1824

The University of Manchester

APT

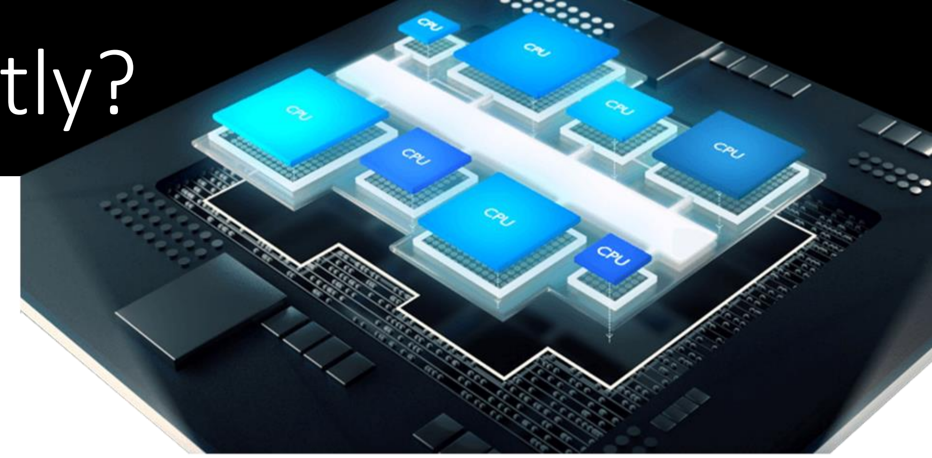
Beyond Polyhedral Analysis of OpenStream Programs

Nuno Miguel Nobre nunomiguel.nobre@manchester.ac.uk

Joint work with: Andi Drebes, Graham Riley and Antoniu Pop

IMPACT 2019: January 23, 2019 | Valencia, Spain

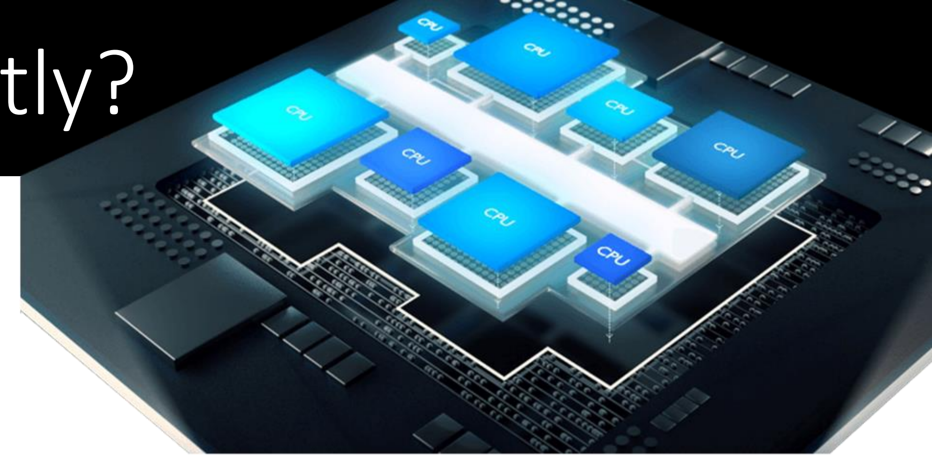
How to exploit today's machines efficiently?



Task-parallel streaming dataflow models
have strong assets:

- Point-to-point synchronization
  - Hide latency
- Numerous opportunities for parallelism
  - Task, data and pipeline
- Scheduling is the runtime's job
- Provide functional determinism

How to exploit today’s machines efficiently?



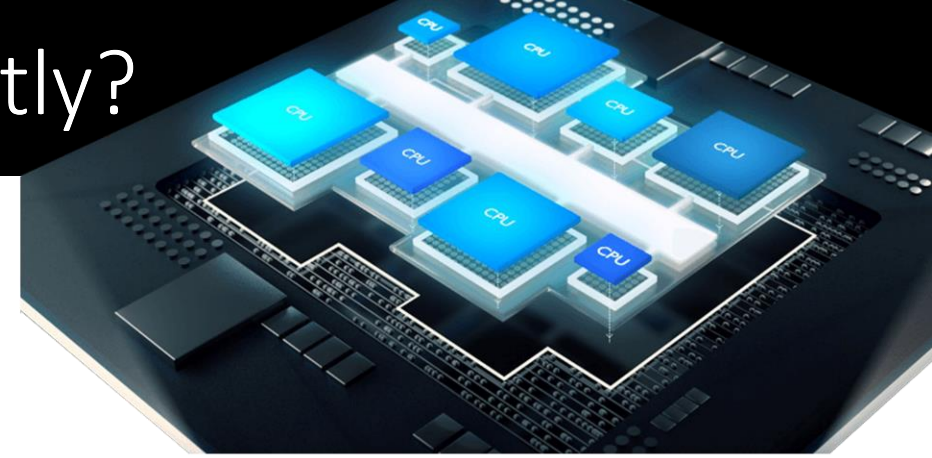
Task-parallel streaming dataflow models have strong assets:

- Point-to-point synchronization
  - Hide latency
- Numerous opportunities for parallelism
  - Task, data and pipeline
- Scheduling is the runtime’s job
- Provide functional determinism

But also disadvantages:

- Manually specified tasks
  - Challenging dependency specification
  - Hard debugging
  - What’s the right granularity?
- Memory footprint: no in-place writes

How to exploit today's machines efficiently?



Task-parallel streaming dataflow models have strong assets:

- Point-to-point synchronization
  - Hide latency
- Numerous opportunities for parallelism
  - Task, data and pipeline
- Scheduling is the runtime's job
- Provide functional determinism

But also disadvantages:

- Manually specified tasks
  - Challenging dependency specification
  - Hard debugging
  - What's the right granularity?
- Memory footprint: no in-place writes

Why the polyhedral model?

- Arbitrarily compose loop transformations inc. tiling $\longrightarrow$ granularity control
- Static program analysis $\longrightarrow$ streams memory footprint/bounding
- Multi-objective: parallelism, **vectorization**, **multi-level cache reuse**
- Compact program representation unlike graph algorithms
- Despite restrictions: **stencils**, dense linear algebra and image filters

Outline

1) Manual granularity tuning
   - Motivating example: Gauss-Seidel stencil

2) Stream bounding & automatic granularity tuning
   - The polynomial indexing problem
   - Future work solutions

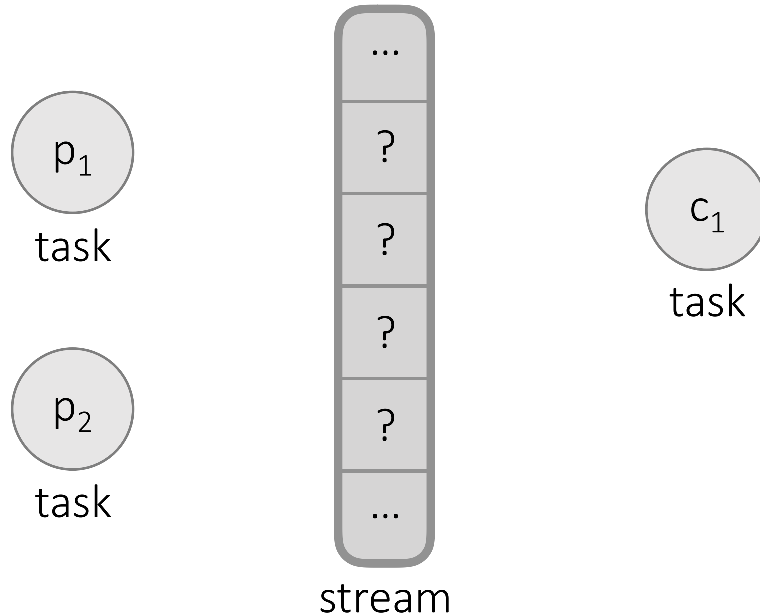
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

} created dynamically at runtime

Tasks access stream elements through sliding **windows**:



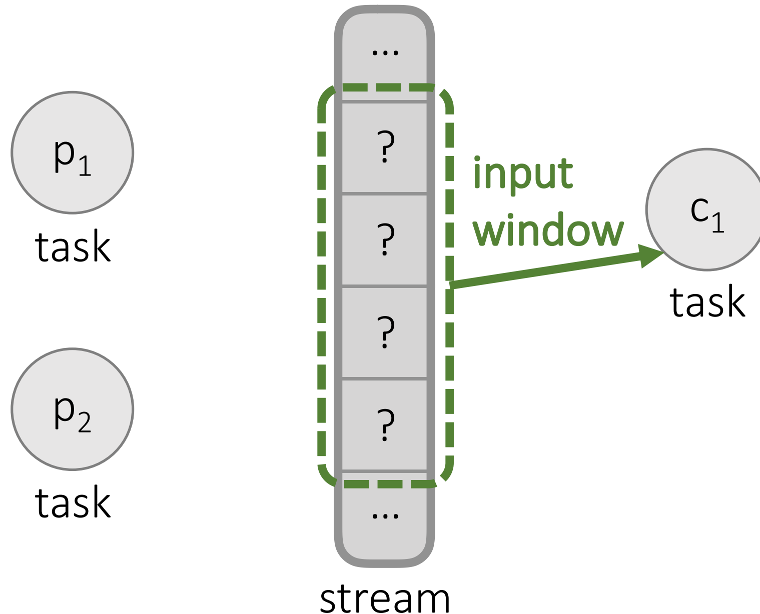
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

} created dynamically at runtime

Tasks access stream elements through sliding **windows**:



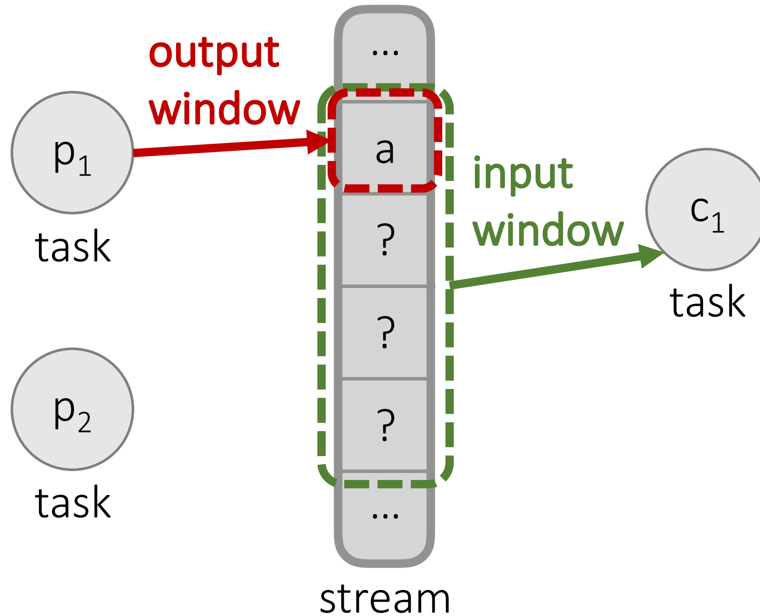
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

} created dynamically at runtime

Tasks access stream elements through sliding **windows**:



Stream accesses dictate the dependencies between tasks

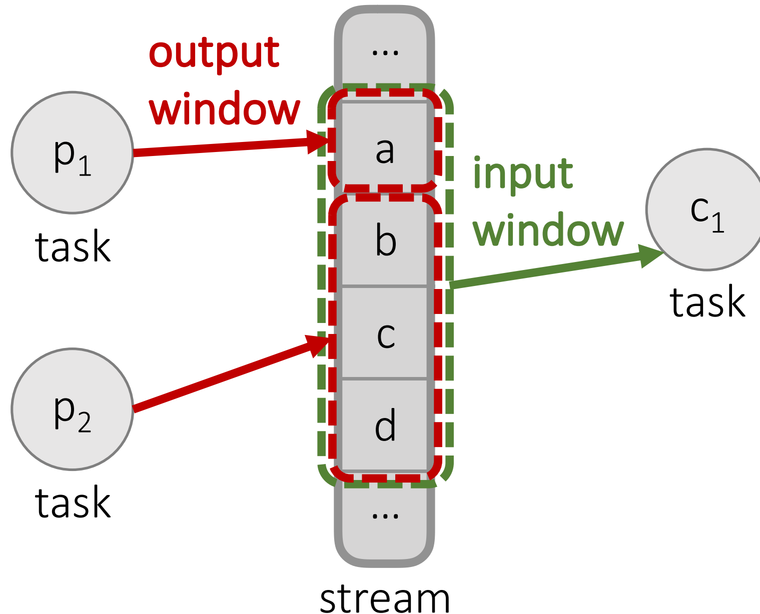
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

} created dynamically at runtime

Tasks access stream elements through sliding **windows**:



Stream accesses dictate the dependencies between tasks

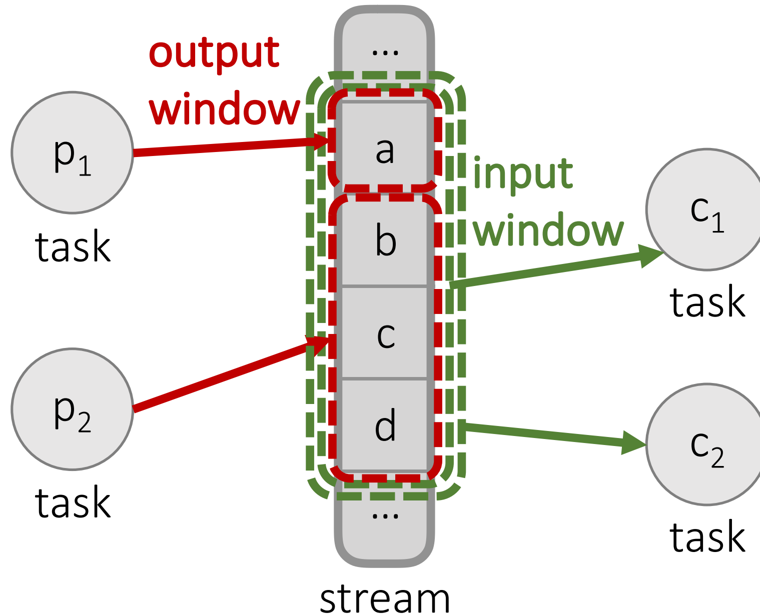
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

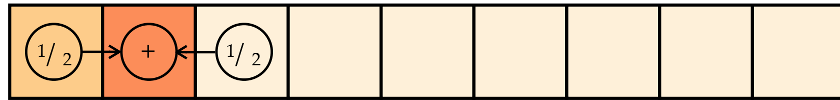
} created dynamically at runtime

Tasks access stream elements through sliding **windows**:



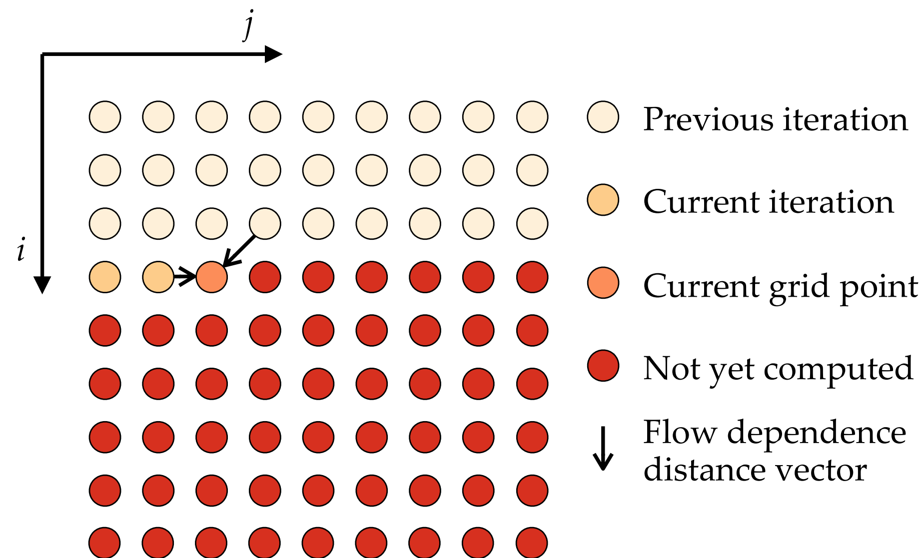
Stream accesses dictate the dependencies between tasks

1D Gauss-Seidel: stencil code granularity tuning



Sequential C [SeqC]

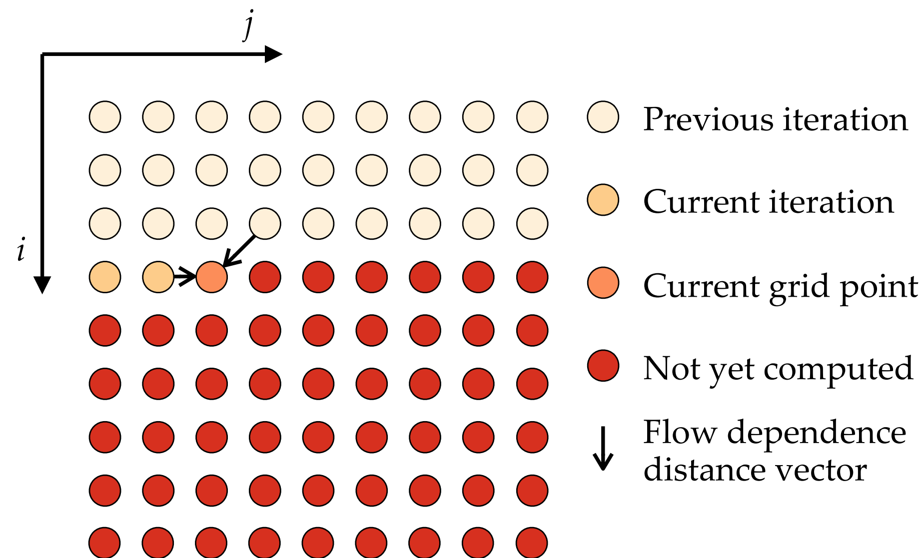
```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```



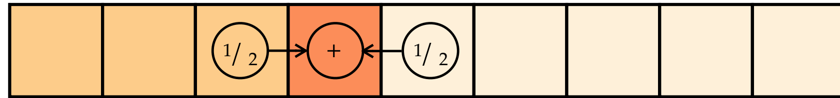
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

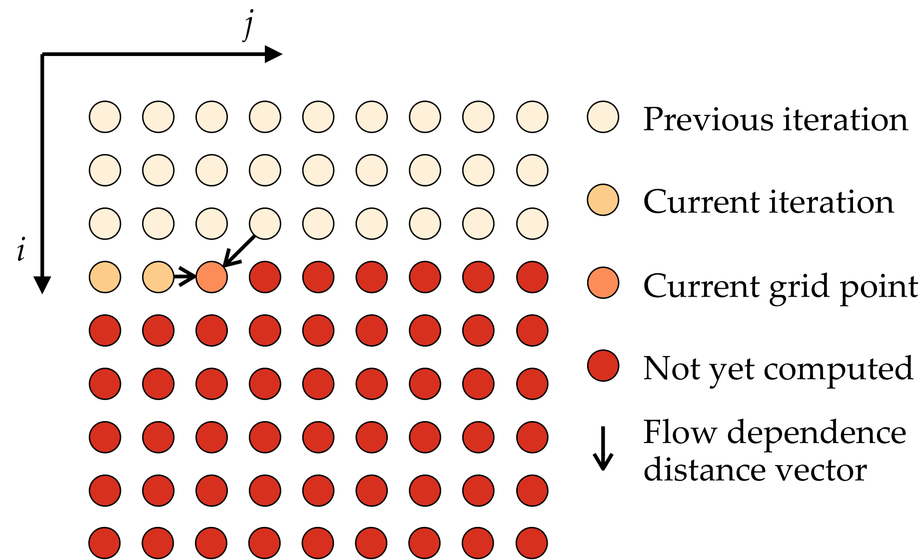


1D Gauss-Seidel: stencil code granularity tuning

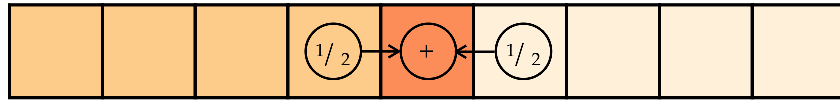


Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

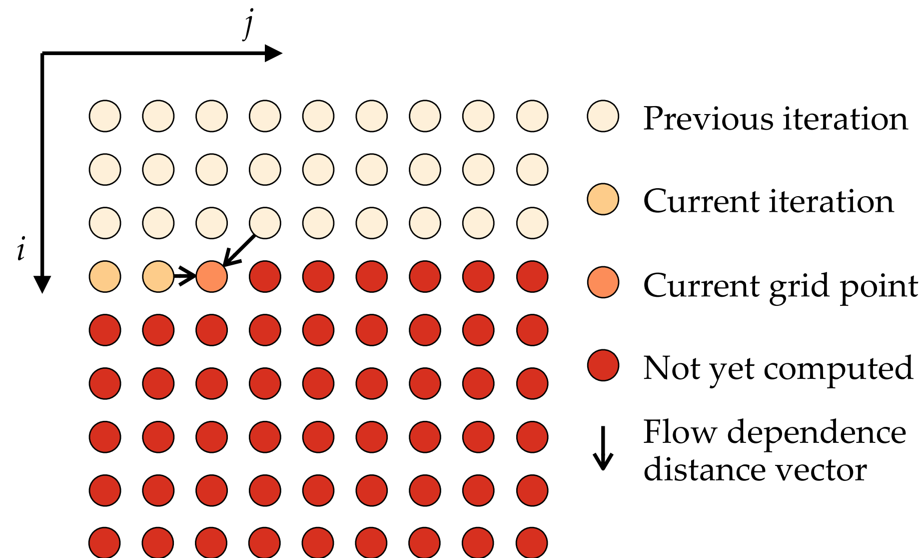


1D Gauss-Seidel: stencil code granularity tuning

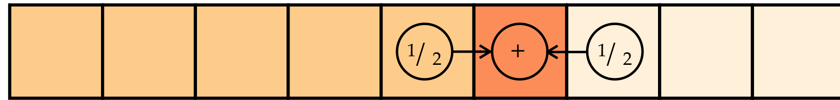


Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

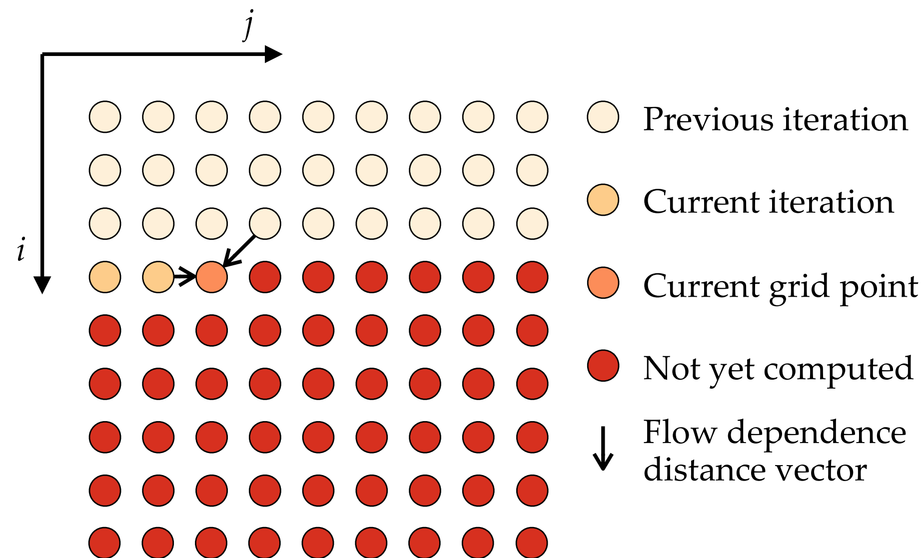


1D Gauss-Seidel: stencil code granularity tuning

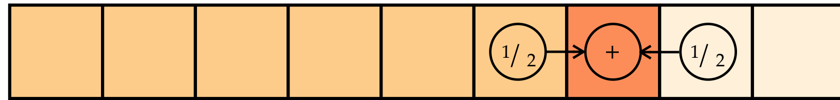


Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

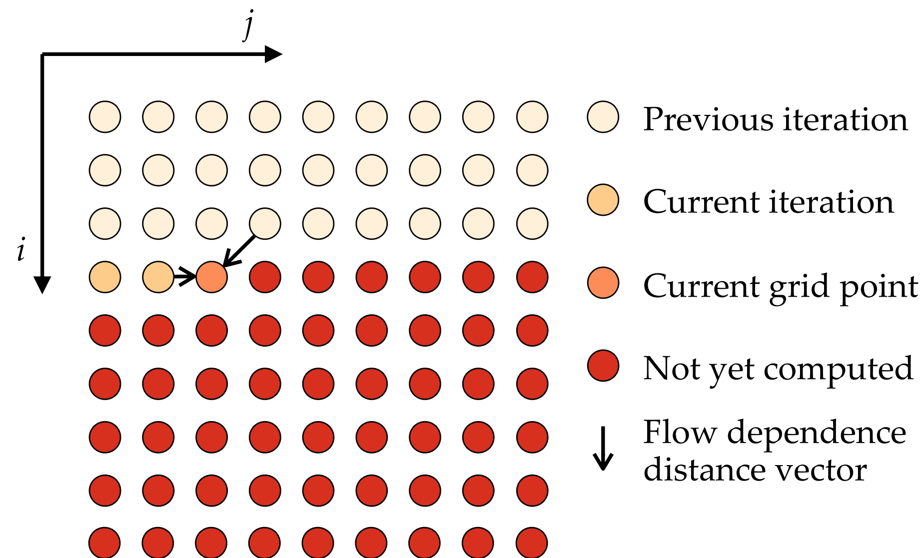


1D Gauss-Seidel: stencil code granularity tuning

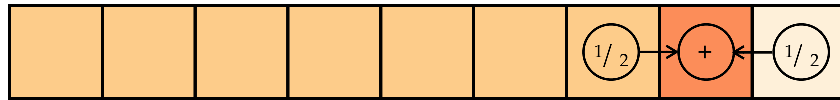


Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

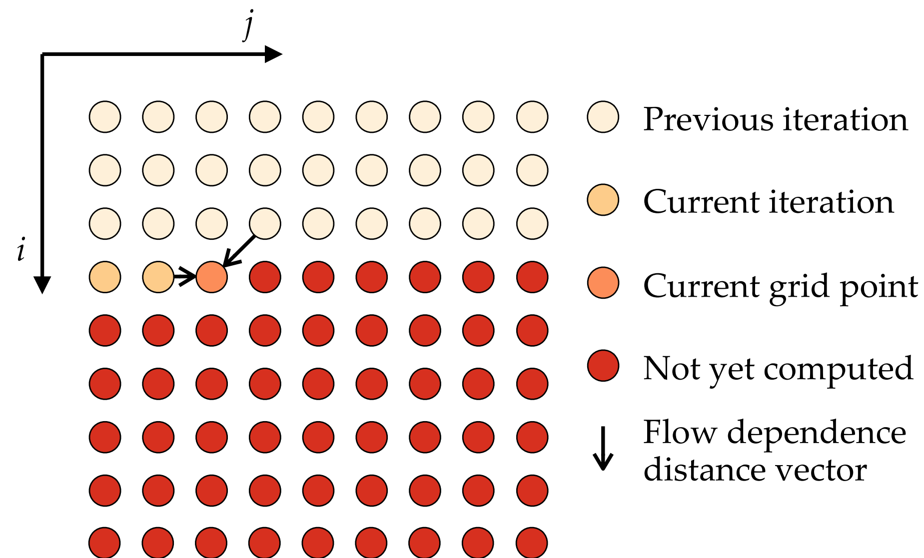


1D Gauss-Seidel: stencil code granularity tuning

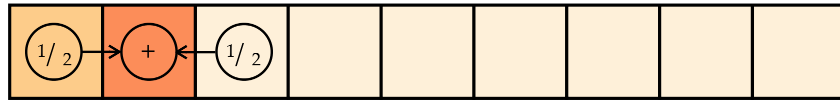


Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

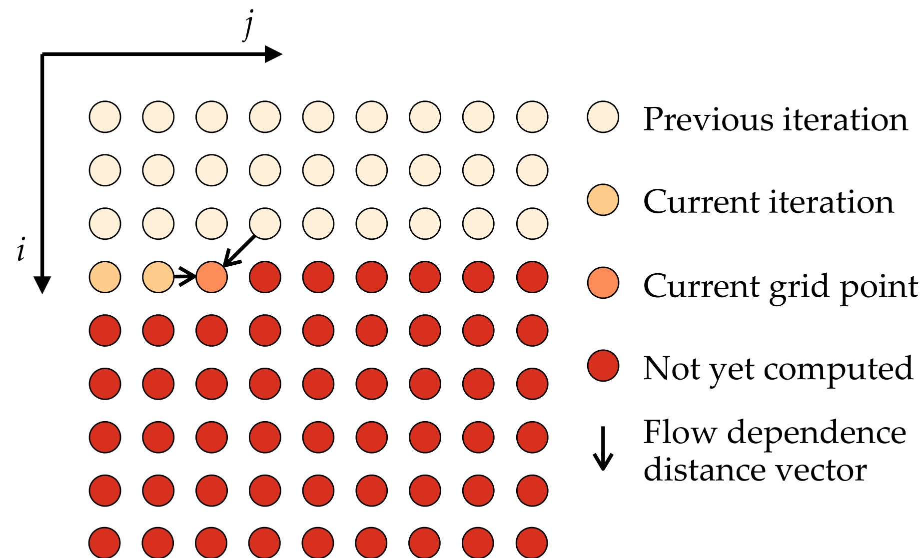


1D Gauss-Seidel: stencil code granularity tuning

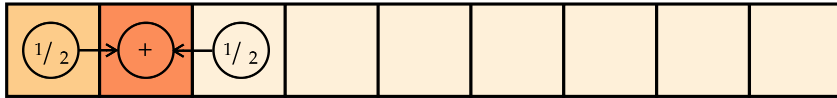


Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

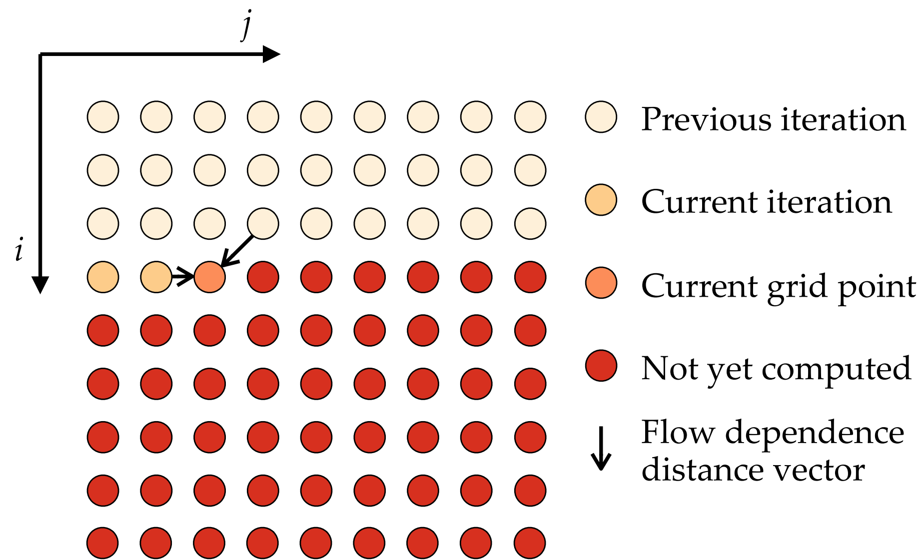


1D Gauss-Seidel: stencil code granularity tuning



Sequential C [SeqC]

```
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```



OpenStream: Fine-grained tasks [OS-FG]

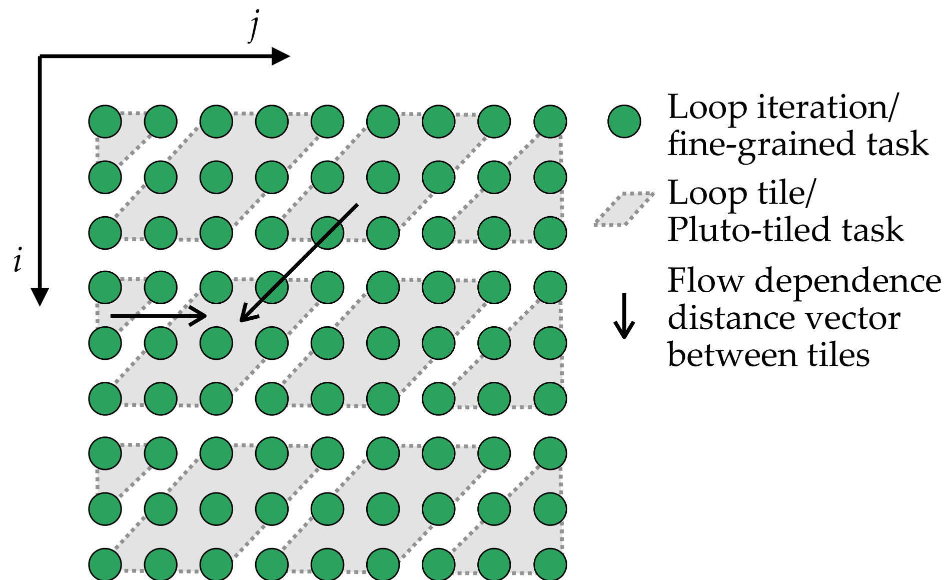
```
stream_array S[N];

for (i = 0; i < I; ++i)
 for (j = 1; j < N - 1; ++j)
  task {
   read once from S[j];     // phi[j] (discarded)
   peek once from S[j - 1]; // phi[j - 1]
   peek once from S[j + 1]; // phi[j + 1]
   write once into S[j];    // phi[j]

   // work function:
   // phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
  }
```

1D Gauss-Seidel: stencil code granularity tuning

1) Semantically equivalent C code (SA)
2) Pluto source-to-source compiler
3) OpenMP parallel code [OMP-PT]
4) OpenStream: Pluto-tiled tasks [OS-PT]



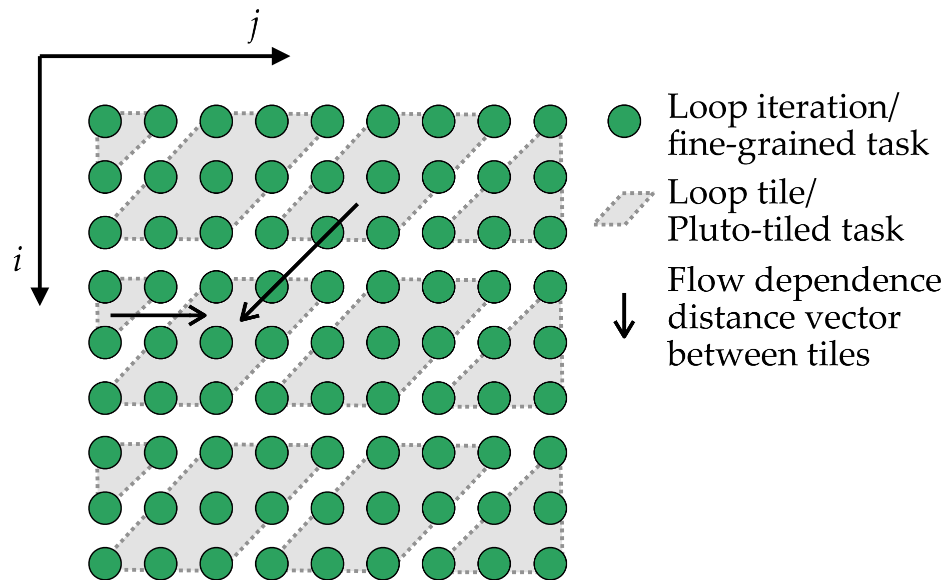
1D Gauss-Seidel: stencil code granularity tuning

1) Semantically equivalent C code (SA)

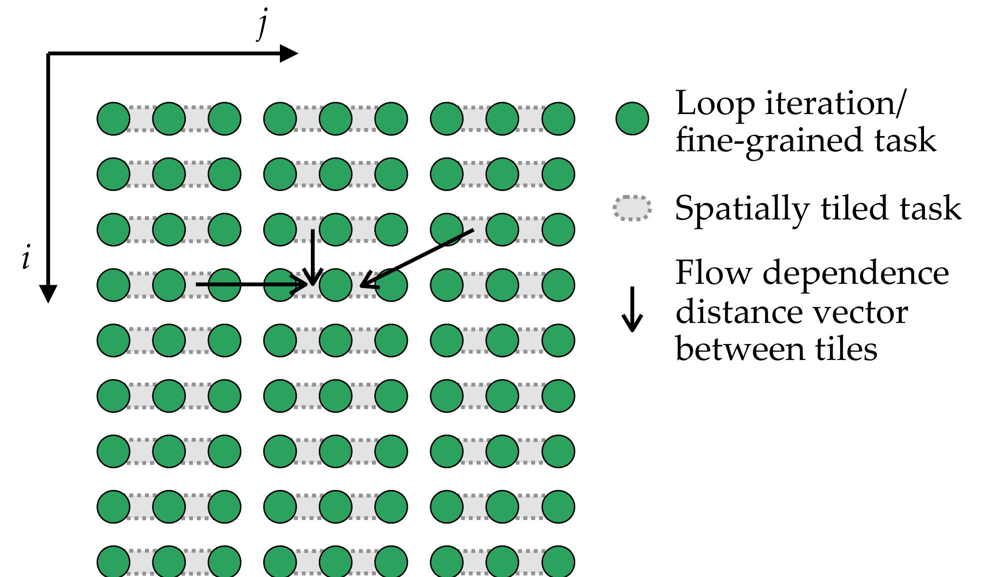
2) Pluto source-to-source compiler

3) OpenMP parallel code [OMP-PT]

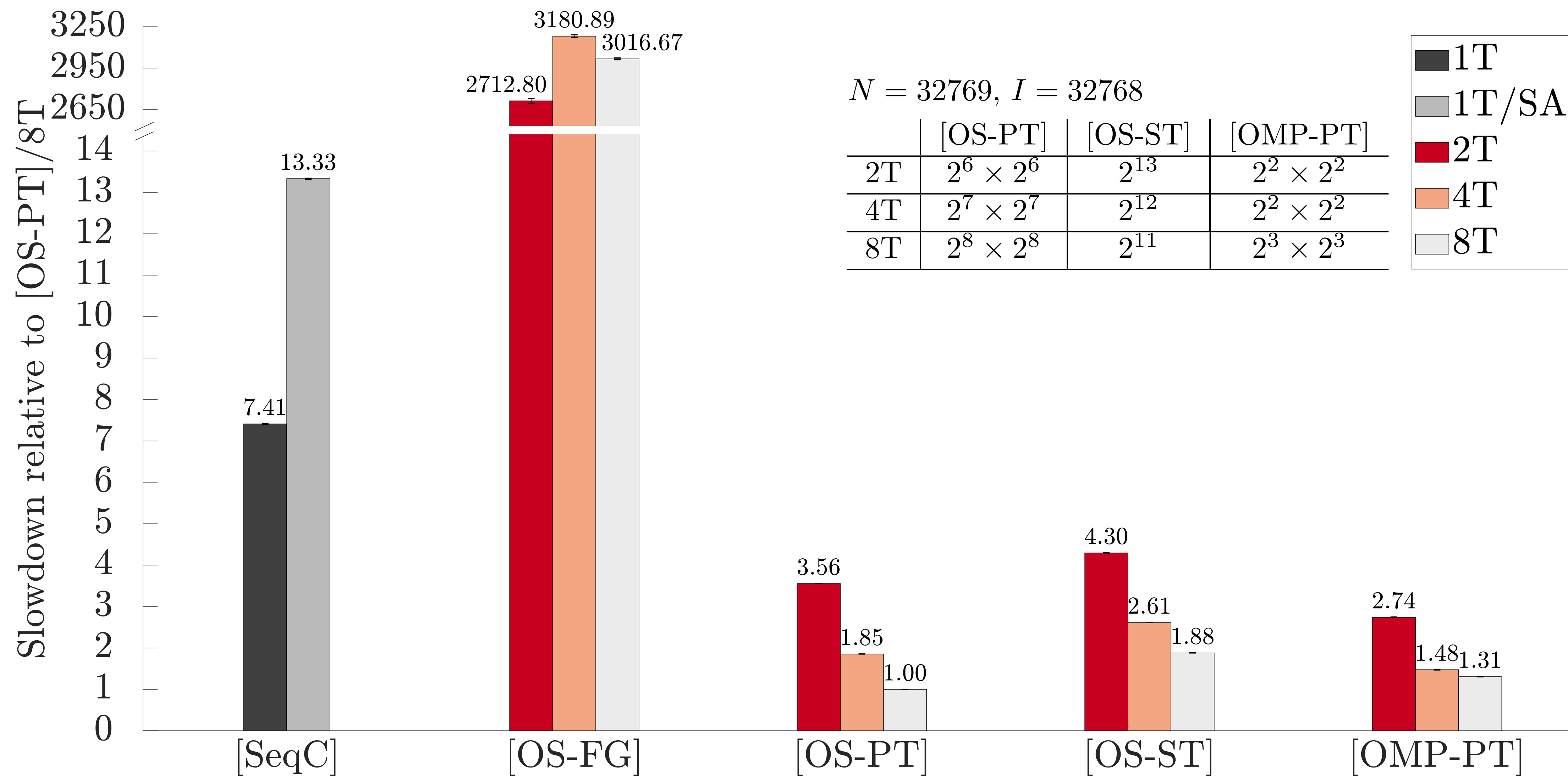
4) OpenStream: Pluto-tiled tasks [OS-PT]



OpenStream: Spatially tiled tasks [OS-ST]



1D Gauss-Seidel: results

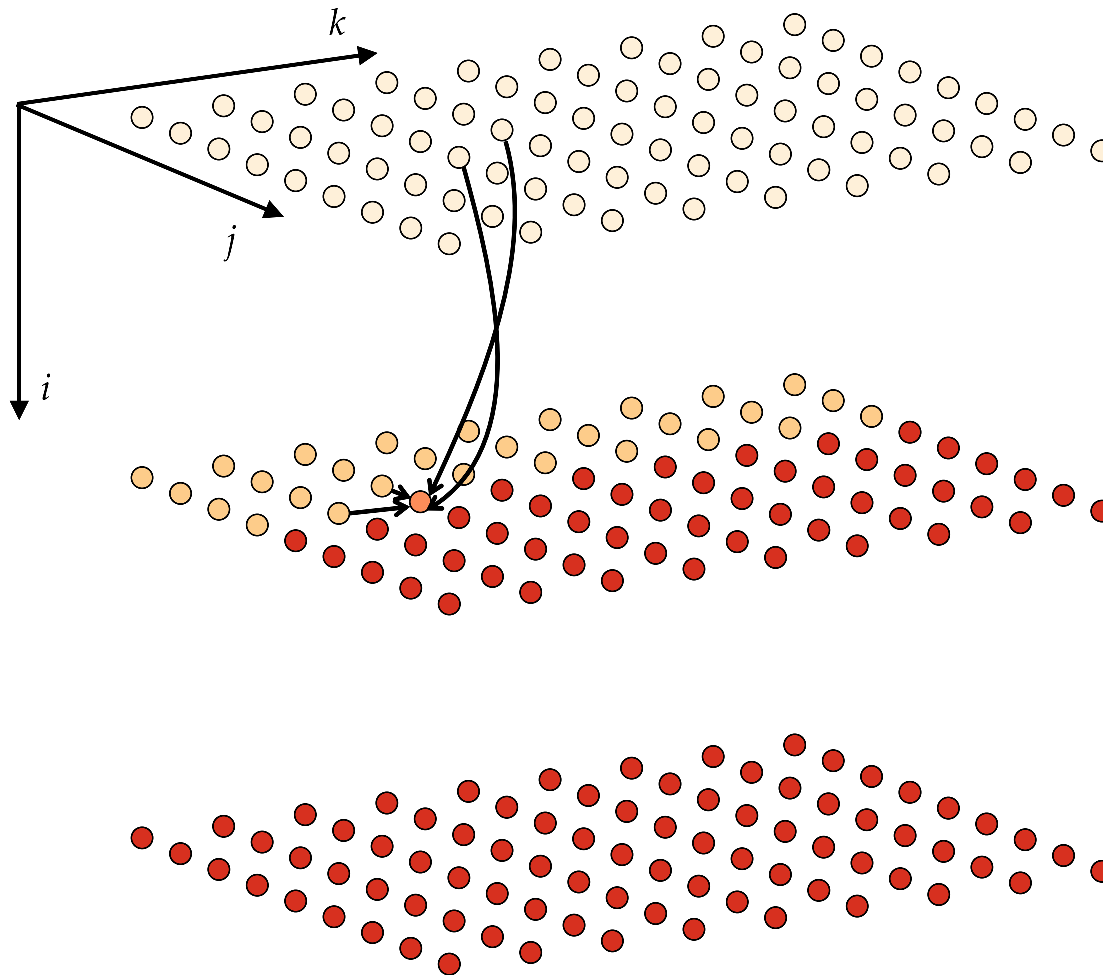


$N = 32769,\ I = 32768$

	[OS-PT]	[OS-ST]	[OMP-PT]
2T	$2^6 \times 2^6$	2^{13}	$2^2 \times 2^2$
4T	$2^7 \times 2^7$	2^{12}	$2^2 \times 2^2$
8T	$2^8 \times 2^8$	2^{11}	$2^3 \times 2^3$

2D Gauss-Seidel: a visual picture

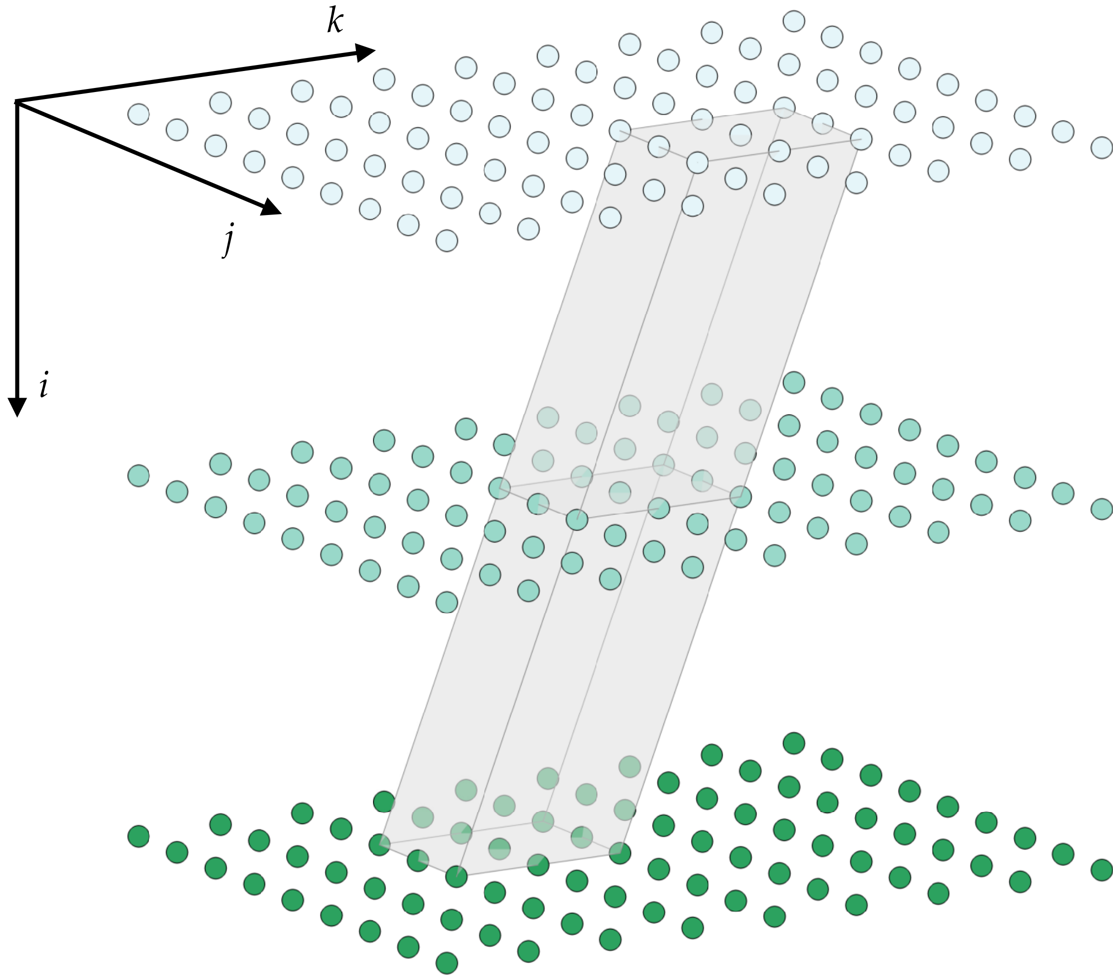
OpenStream: Fine-grained tasks [OS-FG]



- Previous iteration
- Current iteration
- Current grid point
- Not yet computed
- Flow dependence distance vector

2D Gauss-Seidel: a visual picture

OpenStream: Pluto-tiled tasks [OS-PT]



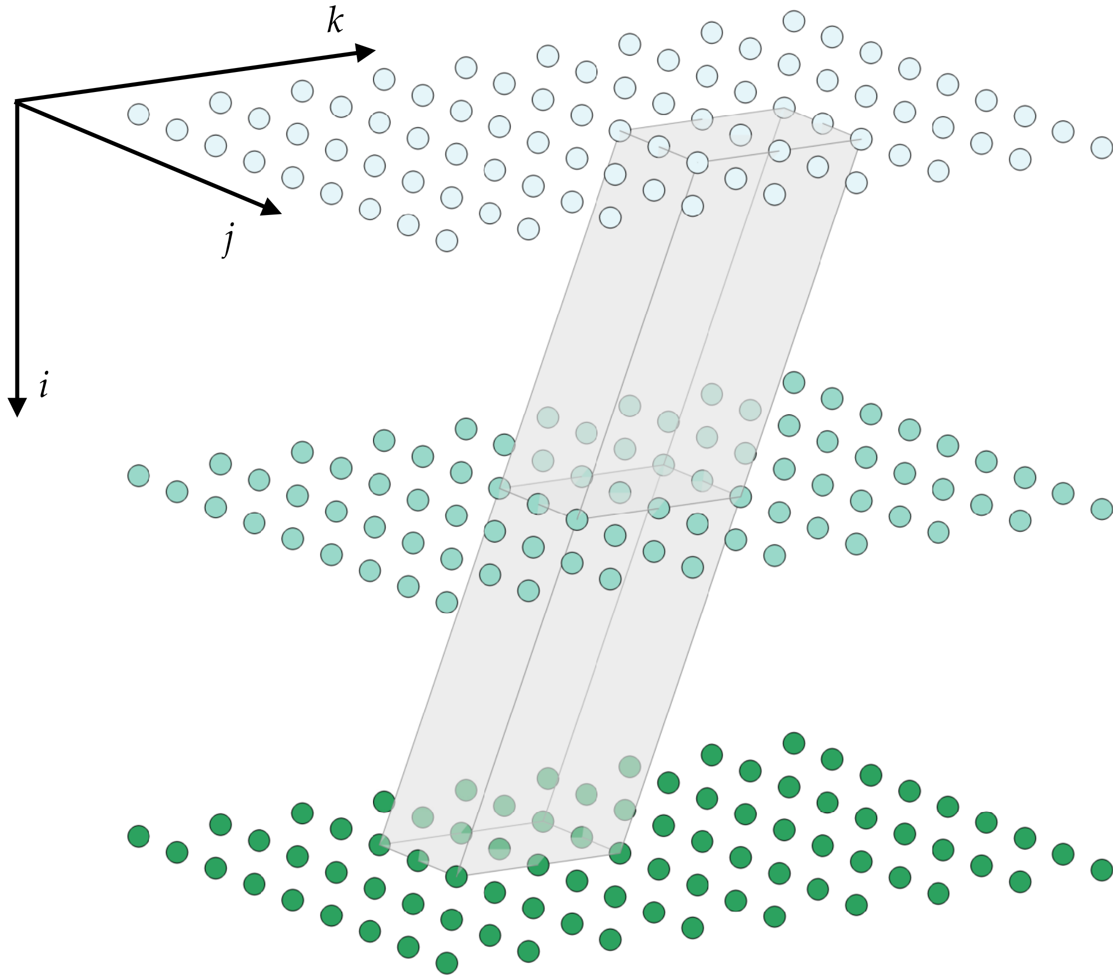
Loop iteration / fine-grained task

Previous iterations of the outer loop

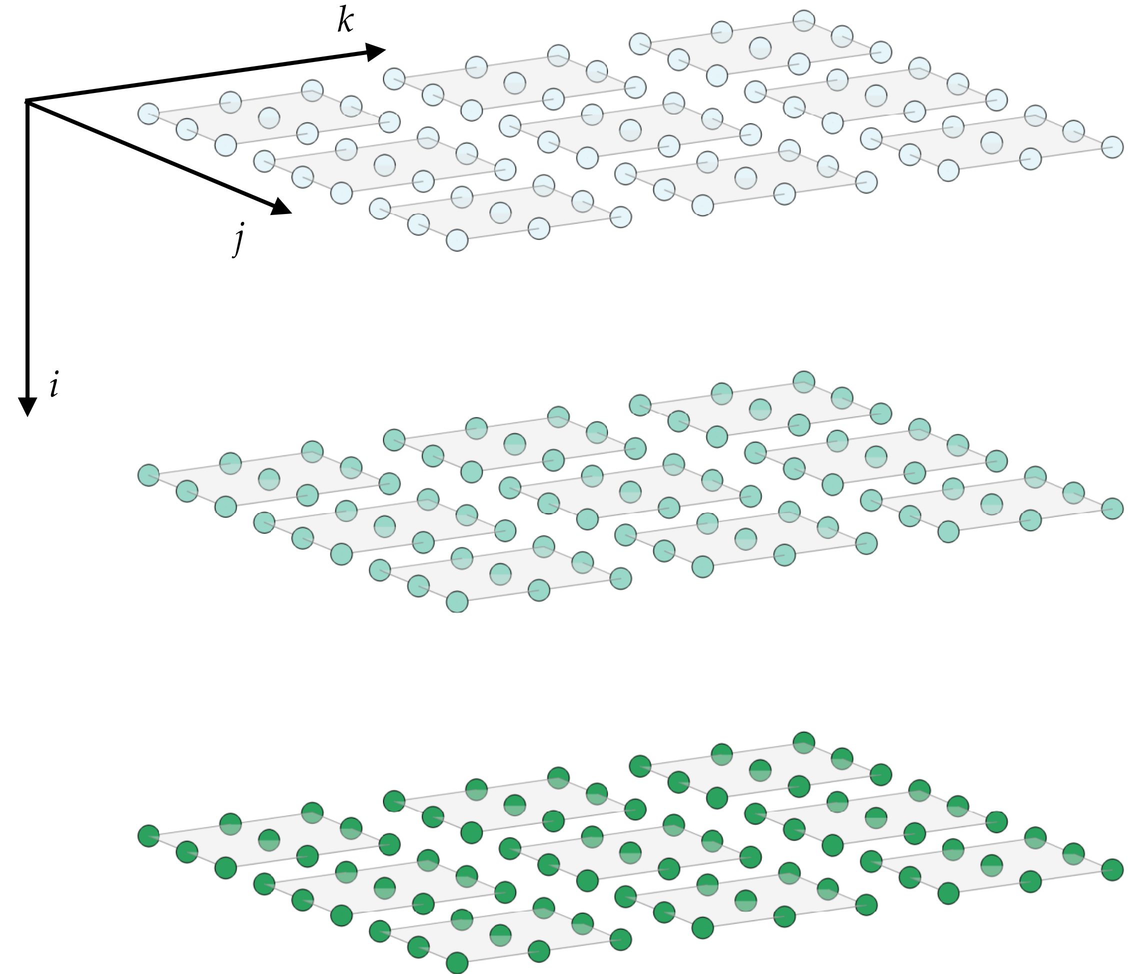
Loop tile / Pluto-tiled task

2D Gauss-Seidel: a visual picture

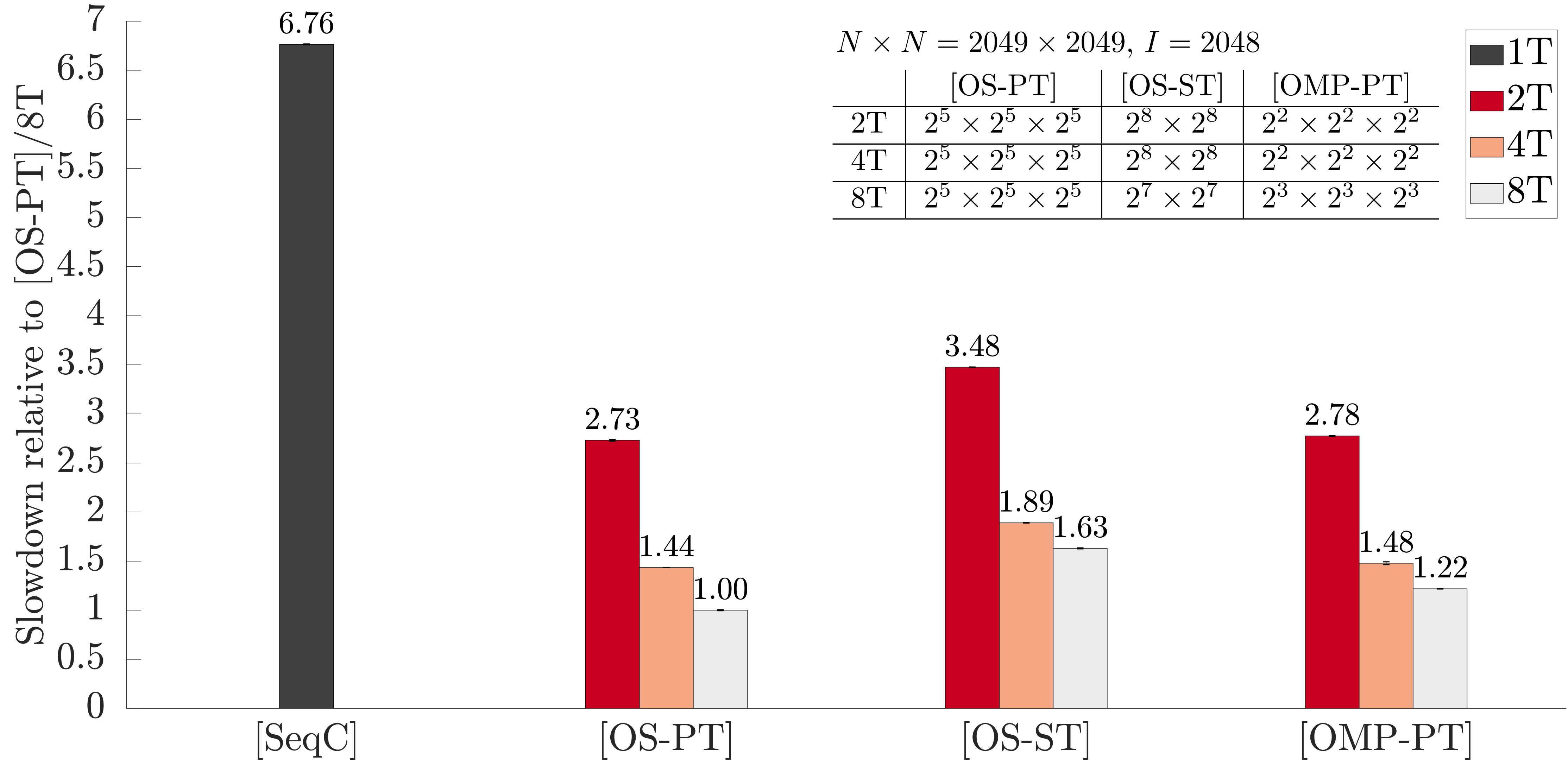
OpenStream: Pluto-tiled tasks [OS-PT]



OpenStream: Spatially tiled tasks [OS-ST]



2D Gauss-Seidel: results



	[OS-PT]	[OS-ST]	[OMP-PT]
2T	$2^5 \times 2^5 \times 2^5$	$2^8 \times 2^8$	$2^2 \times 2^2 \times 2^2$
4T	$2^5 \times 2^5 \times 2^5$	$2^8 \times 2^8$	$2^2 \times 2^2 \times 2^2$
8T	$2^5 \times 2^5 \times 2^5$	$2^7 \times 2^7$	$2^3 \times 2^3 \times 2^3$

The polynomial problem

- Stream indexing is polynomial
  - e.g. parametric tiling

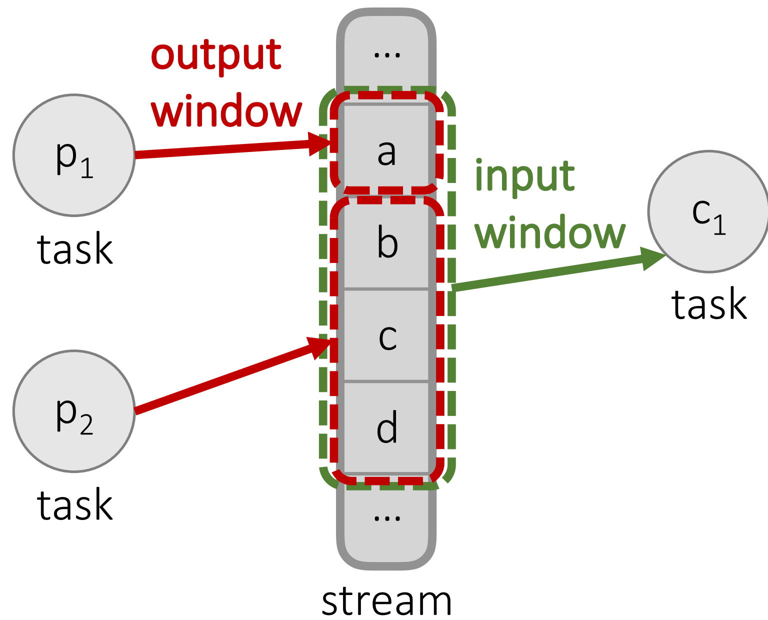
The polynomial problem

- Stream indexing is polynomial
  - e.g. parametric tiling
- Deadlock undecidability
  - Albert Cohen, Alain Darte, and Paul Feautrier. 2016. Static Analysis of OpenStream Programs

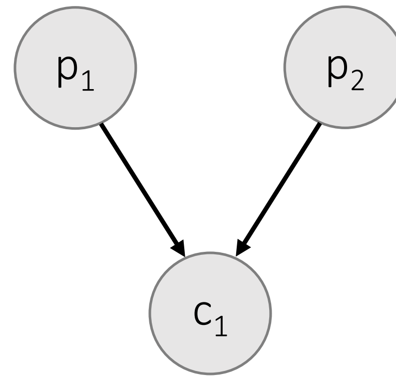
The polynomial problem

- Stream indexing is polynomial
  - e.g. parametric tiling
- Deadlock undecidability
  - Albert Cohen, Alain Darte, and Paul Feautrier. 2016. Static Analysis of OpenStream Programs
- Schedule found: no deadlock
  - Paul Feautrier and Albert Cohen. 2018. On Polynomial Code Generation

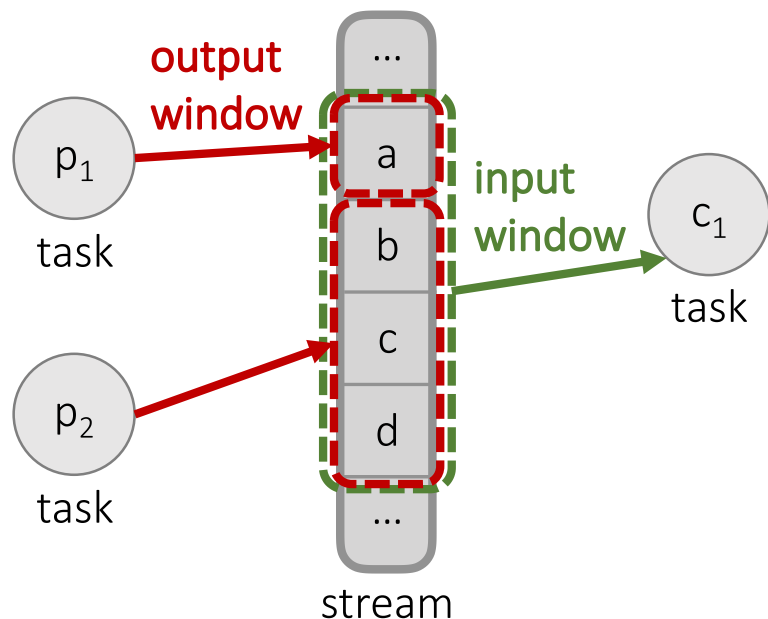
Future work: bounding streams



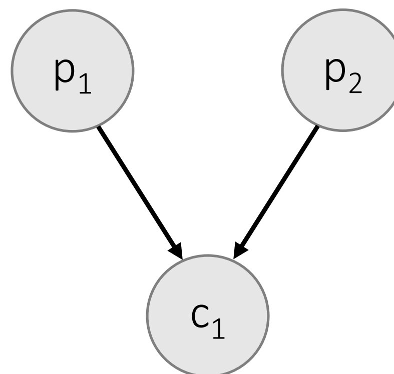
Dataflow task graph



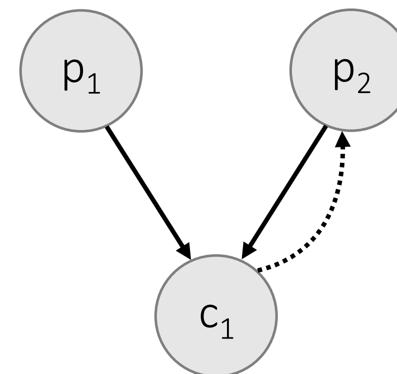
Future work: bounding streams



Dataflow task graph

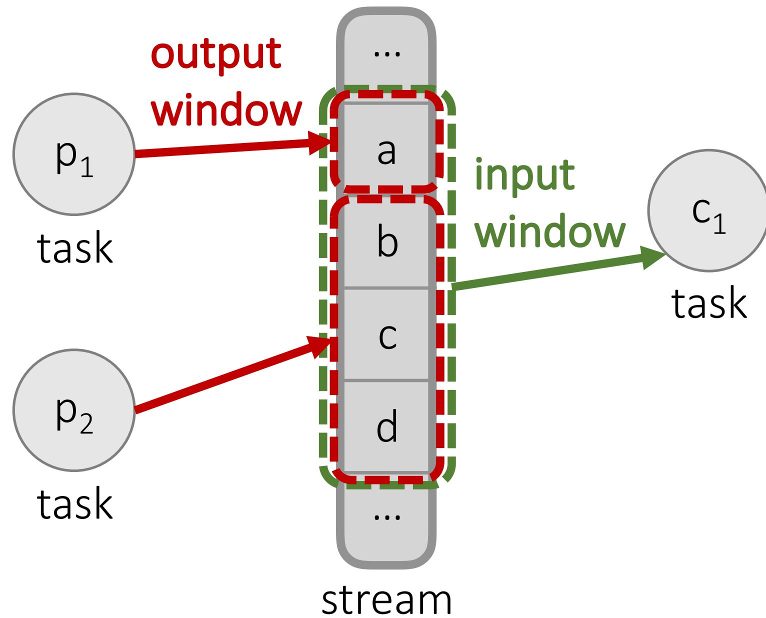


3-element stream:
deadlock

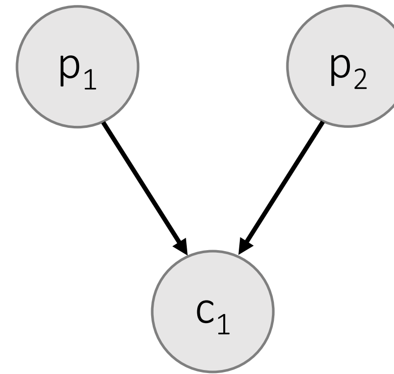


Back-pressure dependencies
- Dataflow task graph: new edges (cycle)
- Poly. model: "just" new schedule restrictions (no schedule)

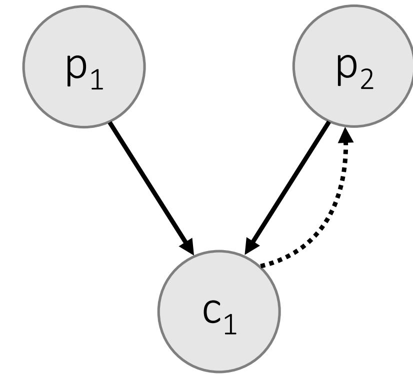
Future work: bounding streams



Dataflow task graph



3-element stream:
deadlock

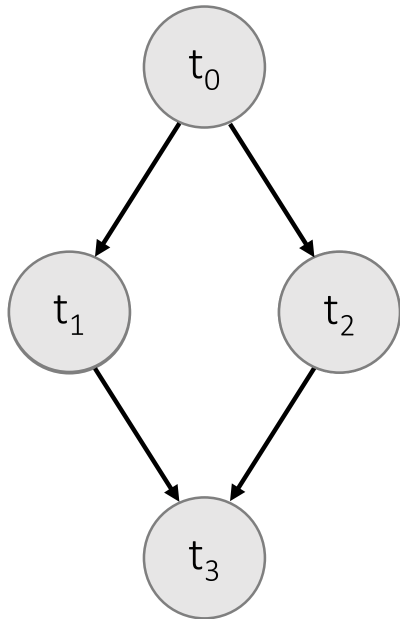


Back-pressure dependencies
- Dataflow task graph: new edges (cycle)
- Poly. model: “just” new schedule restrictions (no schedule)

If schedule found: OpenStream’s runtime can schedule the program

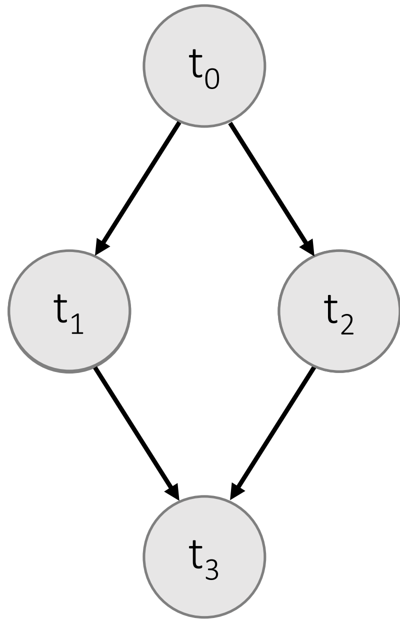
Future work: coarsening task graphs

Dataflow task graph

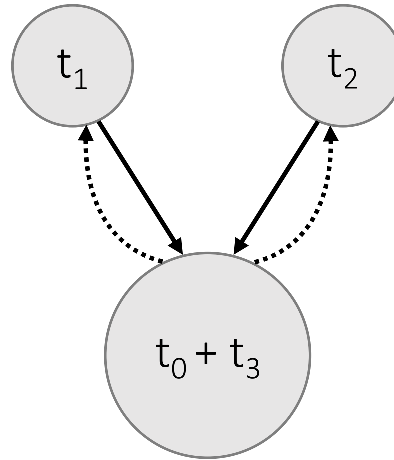


Future work: coarsening task graphs

Dataflow task graph

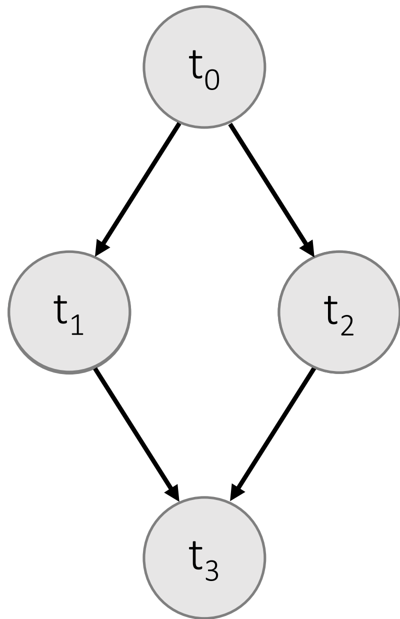


Arbitrary coarsening:
deadlock

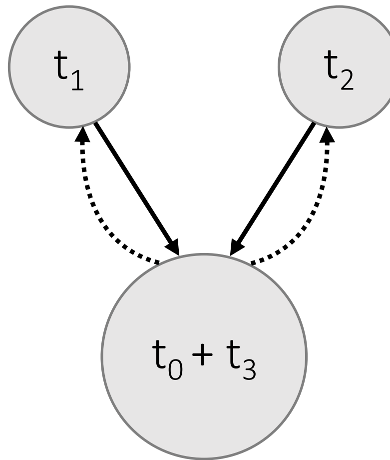


Future work: coarsening task graphs

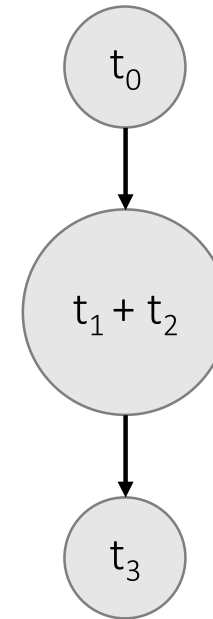
Dataflow task graph



Arbitrary coarsening:
deadlock



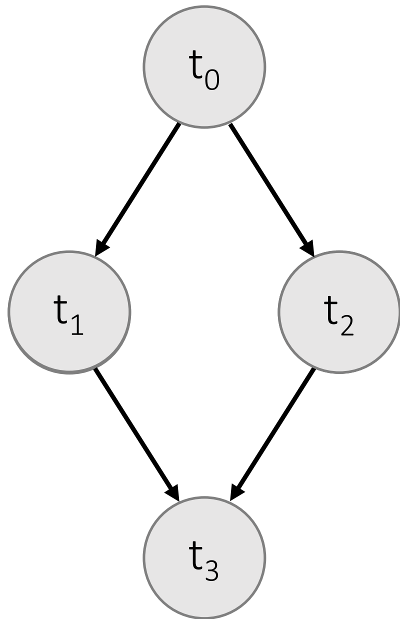
e.g. coalescing instances of the same task



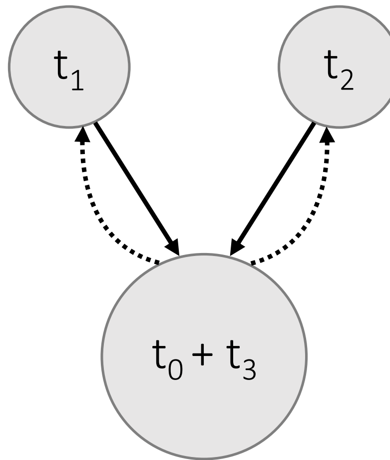
Loop strip-mining, facilitated by stream mushing

Future work: coarsening task graphs

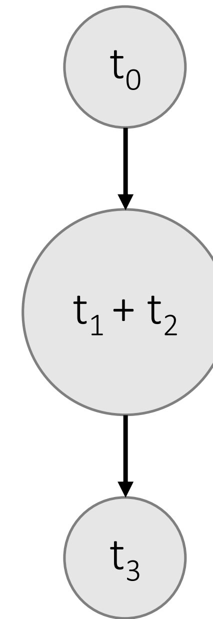
Dataflow task graph



Arbitrary coarsening: deadlock



e.g. coalescing instances of the same task



Loop strip-mining, facilitated by stream mushing

If schedule found: OpenStream's runtime can schedule the program

Summary

- Task-parallel dataflow programs can benefit from polyhedral transformations
- Analyses and transformations are hindered by polynomials
- Bounding streams: adding back-pressure dependencies and finding a schedule
- Granularity control: loop strip-mining? how do we align this w/ current techniques?