The Swan Approach to Task Dataflow-Style Execution

Hans Vandierendonck

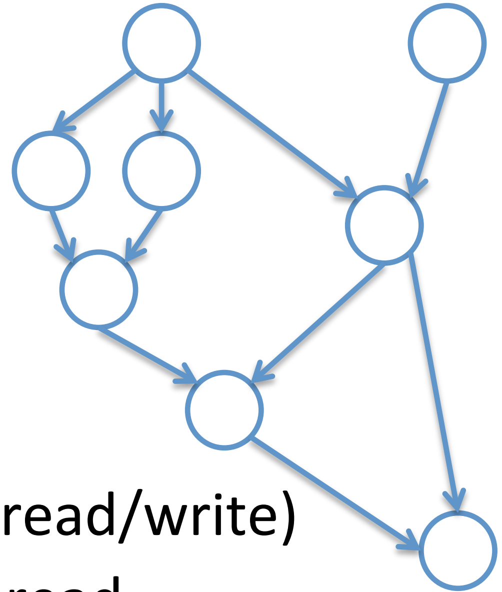
Queen's University Belfast

Work performed at FORTH-ICS, Greece

Queen's University Belfast

Task Dataflow Parallelism: What?

- A program as a collection of inter-dependent tasks
  - Tasks are ordered as a DAG
  - Tasks are ready to execute when the tasks that they depend on have finished



- Out-of-order execution
  - Programmer annotates memory footprint of task and its access mode (read/write)
  - Task graph generated by sequential thread
  - Dependencies inferred from annotations
  - Cf. out-of-order execution in superscalar processors

Queen's University Belfast

Task Dataflow Parallelism: Why?

- Deal with parallel programming issues
  - Determinism, debugging, complexity
  - Limited programming overhead
- Applications
  - High-performance computing – “lookahead”
    - StarSS (SMPSS, CellSS), StarPU, SuperMatrix, codelets, …
  - Wavefront computations
    - h264 video encoding/decoding [Chi & Juurlink, ICS’11]
    - Smith-Waterman [Agrawal et al, IPDPS’10]
  - Concurrent collections [Budimlic et al, Sci. Prog. ’10]
  - OoOJava [Jenista et al, PPoPP’10]
  - Legions [Aitken et al, SC’12]

Queen's University Belfast

Design Considerations

- DAG dependency tracking can be very costly
  - Memory footprint matching & partially overlapping arguments
  - Unknown DAG branching factor
- Asymmetric schedulers
  - *Master thread* executes DAG-generating procedure
  - *Worker threads* execute tasks in DAG
  - Single level of parallelism
  - E.g. SMPSS, StarPU, SuperMatrix
- Recursive or divide-and-conquer parallelism
  - Remains best known way to construct many algorithms
  - Generate deep spawn trees
- Mixing recursive parallelism and task dataflow?

Contributions and Overview

- Efficient dependency tracking
  - versioned objects
  - with tickets, without edges in the DAG
- Extend Cilk scheduler
  - to unify recursive parallelism with dataflow parallelism
- Experimentally evaluate performance
  - on par with Cilk++, outperforms SMPSS

Contributions and Overview

- Efficient dependency tracking
  - versioned objects
  - with tickets, without edges in the DAG
- Extend Cilk scheduler
  - to unify recursive parallelism with dataflow parallelism
- Experimentally evaluate performance
  - on par with Cilk++, outperforms SMPSS

Queen's University Belfast

Versioned Objects: Adding Task Dataflow to Cilk++

- Versioned objects

  versioned<T> obj;

- Argument annotations

  indep<T> *read-only*

  outdep<T> *read/write but no exposed reads*

  inoutdep<T> *read/write*

  cinoutdep<T> *commutative*

  reduction<M> *reduction*

  -- T is a C++ type

  -- M is a C++ structure describing the monad with type T, an identity value and a reduction operator

- Independent fork/join

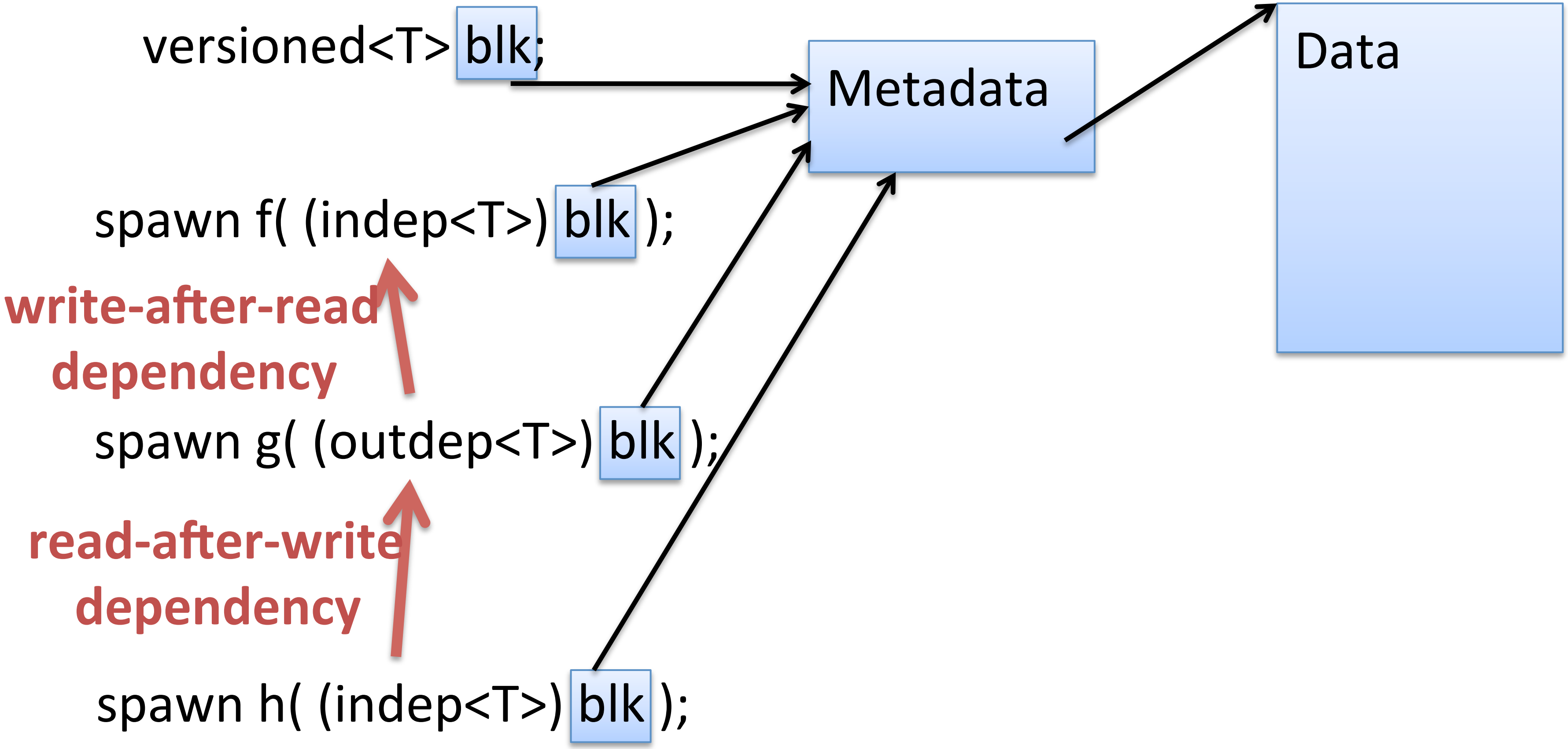
```
int x;
spawn f(x);
…
sync;
```

- Dependency-aware fork/join

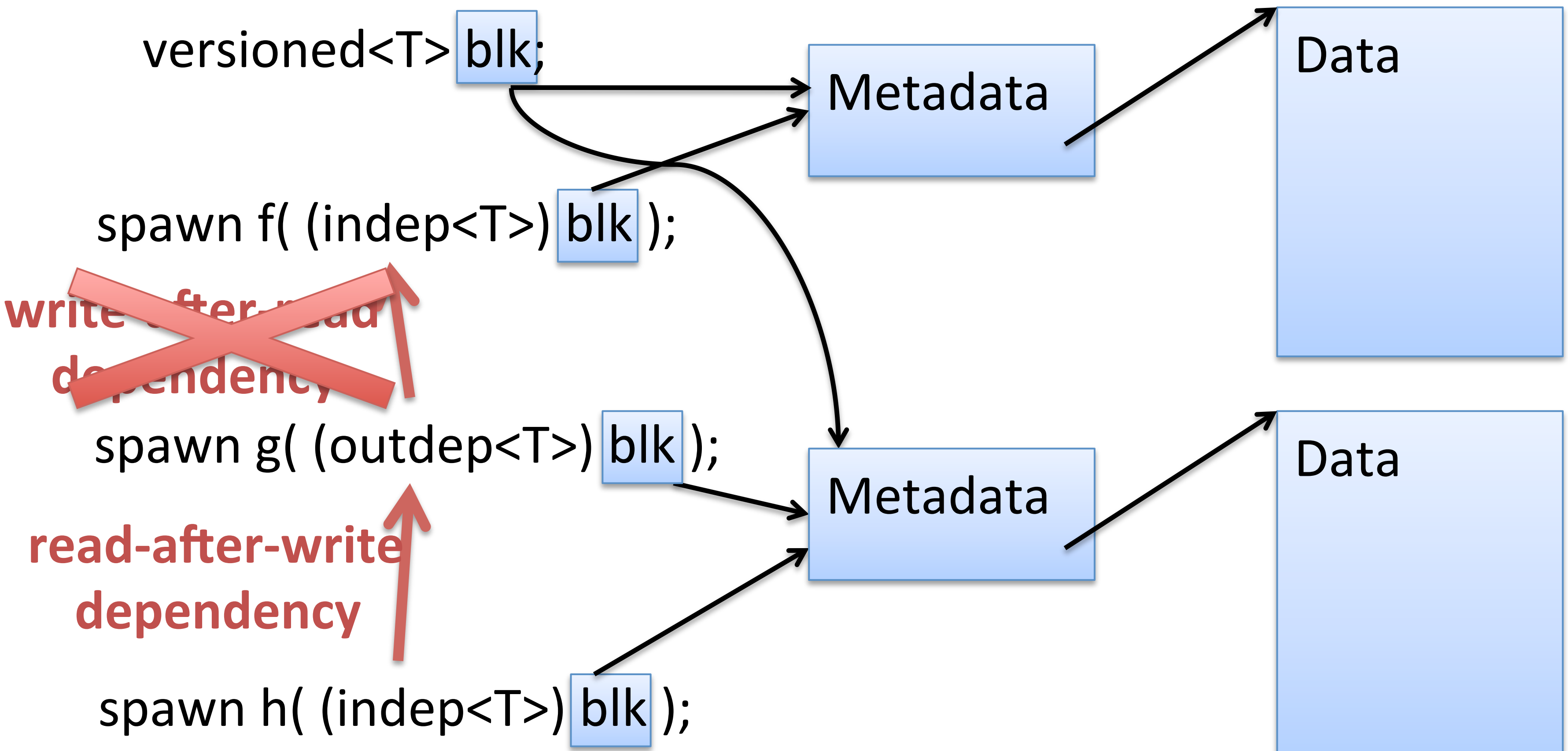
```
versioned<int> x;
spawn f( (indep<int>)x );
…
sync;
```

- Retain implicit sync at end of procedure

Versioned Objects: Fast Metadata Determination



Versioned Objects: Versioning (a.k.a. Renaming)



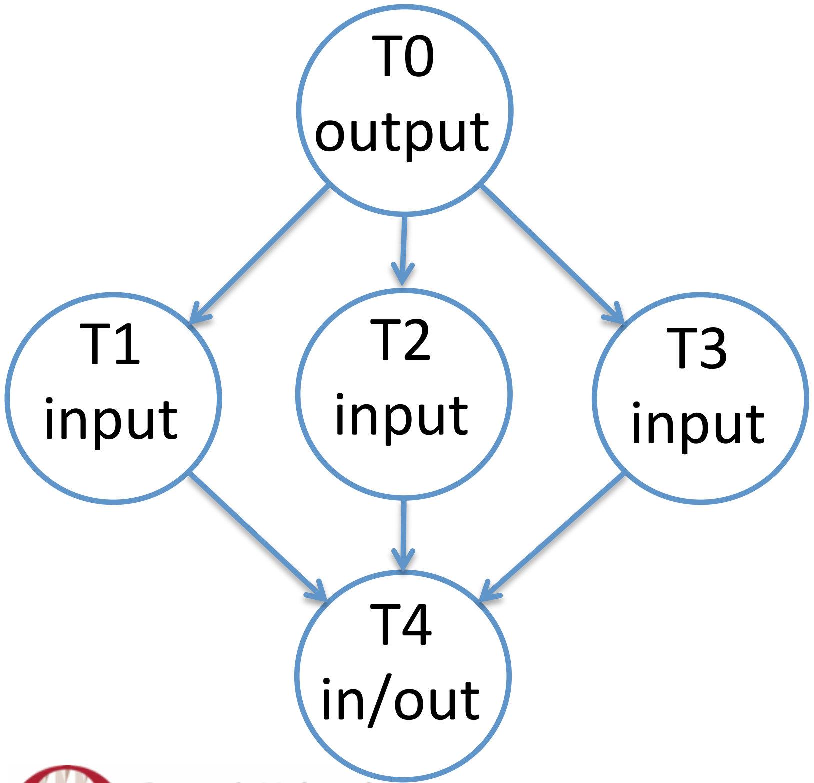
Contributions and Overview

- Efficient dependency tracking
  – versioned objects
  – with tickets, without edges in the DAG
- Extend Cilk scheduler
  – to unify recursive parallelism with dataflow parallelism
- Experimentally evaluate performance
  – on par with Cilk++, outperforms SMPSS

Queen's University Belfast

Observation: The DAG Is a Hypergraph

A sequence of tasks T0, T1, …, T4 (one argument) and its DAG

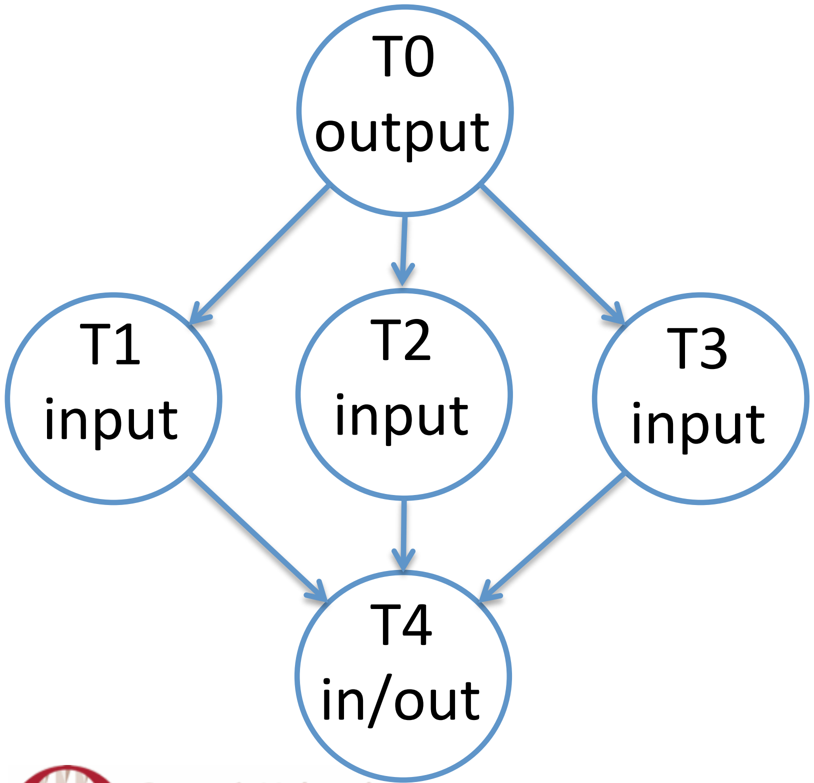


A hypergraph is:

- A graph where edges connect sets of nodes

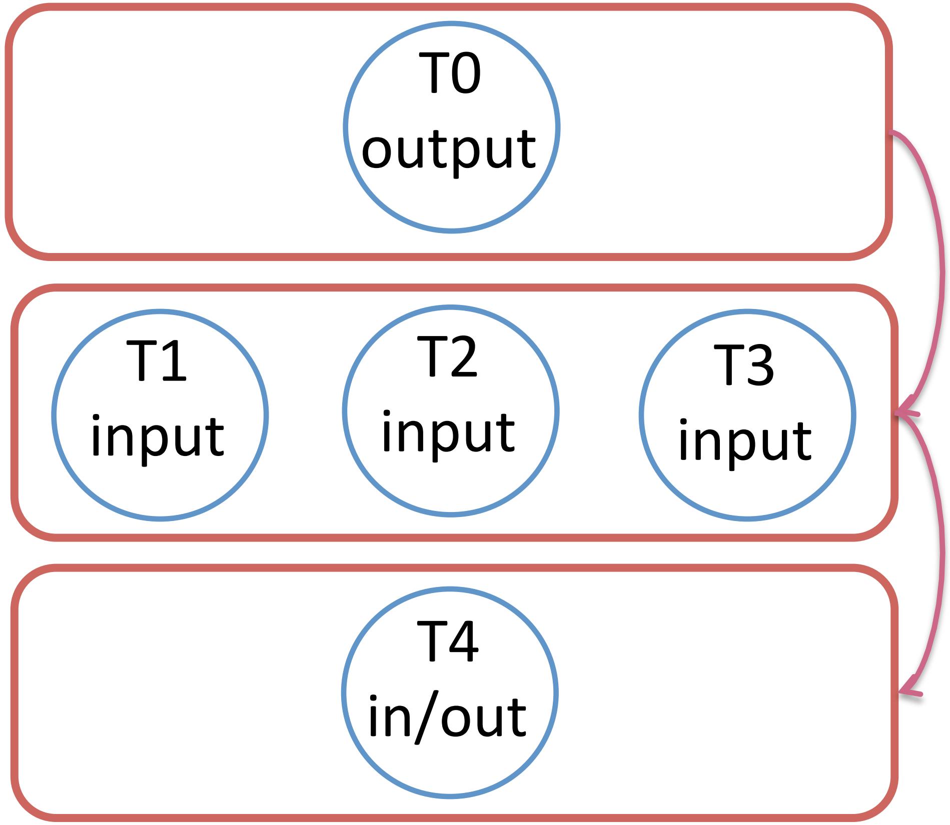
Observation: The DAG Is a Hypergraph

A sequence of tasks T0, T1, …, T4 (one argument) and its DAG



… is actually a hypergraph

… is a sequence of groups of tasks



Queen's University Belfast

Dataflow Synchronization through Tickets

- Ticket locks
  - – Fair queuing of customers
  - – One global counter
  - – One next counter

	Enqueue	Ready?	Dequeue
actions	ticket := next++	ticket = global	++global

No. of arrived customers

No. of served customers

Tickets

Version for in, out and inout deps

- Two queues: readers, writers
  - *Readers may go in parallel, writers execute in isolation*
  - Readers wait on all older writers
  - Writers wait on all older readers and all older writers

	Enqueue	Ready?	Dequeue
input (reader)	++R.next w := W.next	w = W.global	++R.global
output (writer)	if R.next != R.global or W.next != W.global then rename() ++W.next	true	++W.global
in/out (reader /writer)	r := R.next++ w := W.next++	r = R.global and w = W.global	++R.global ++W.global

Queen's University Belfast

Ticket-Based Dependency Tracking

Metadata

R.next = 0
R.global = 0
W.next = 0
W.global = 0

T0
output

Ticket-Based Dependency Tracking

Metadata

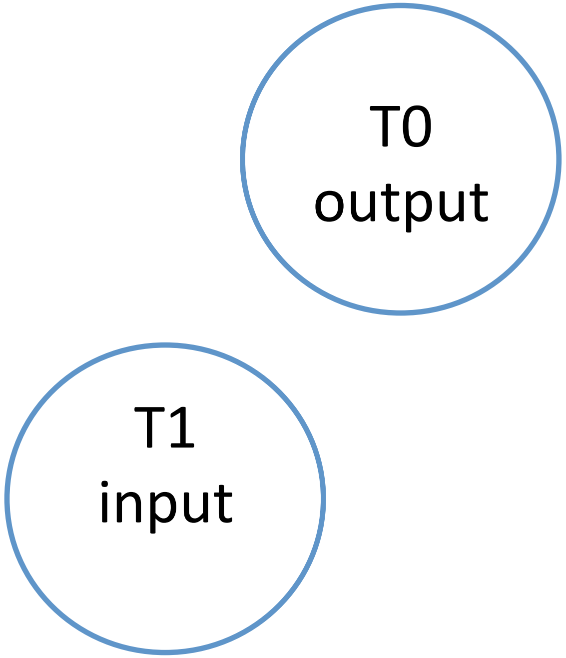
R.next = 0
R.global = 0
W.next = 1
W.global = 0

T0
output

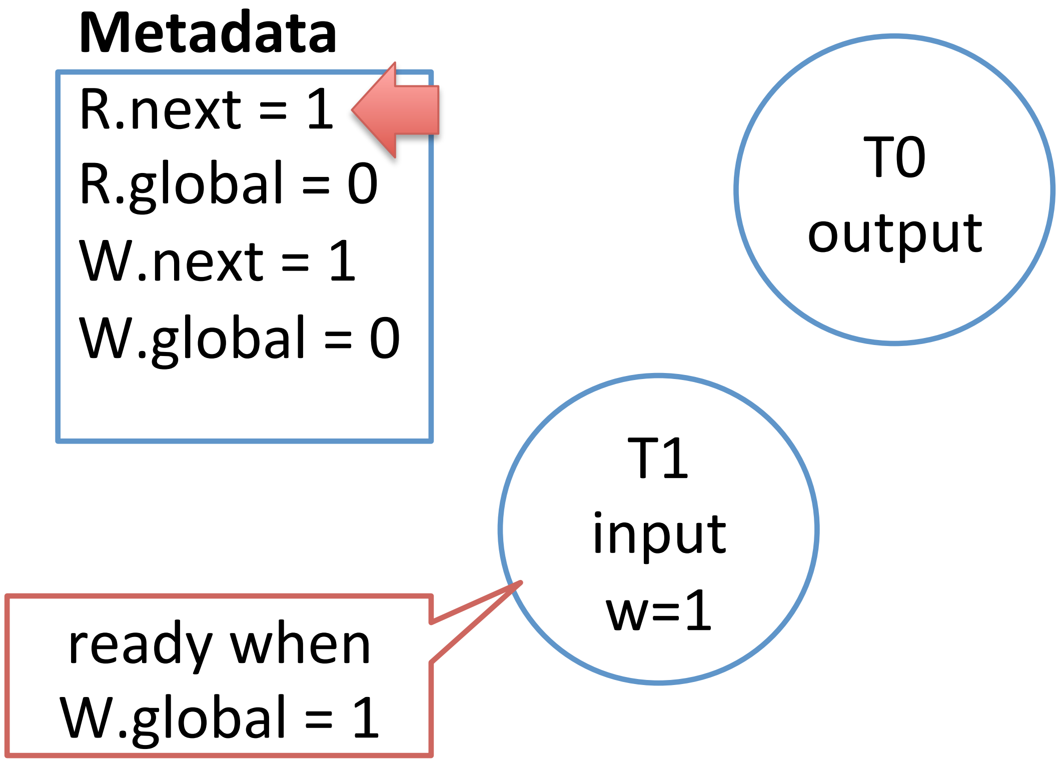
Ticket-Based Dependency Tracking

Metadata

R.next = 0
R.global = 0
W.next = 1
W.global = 0



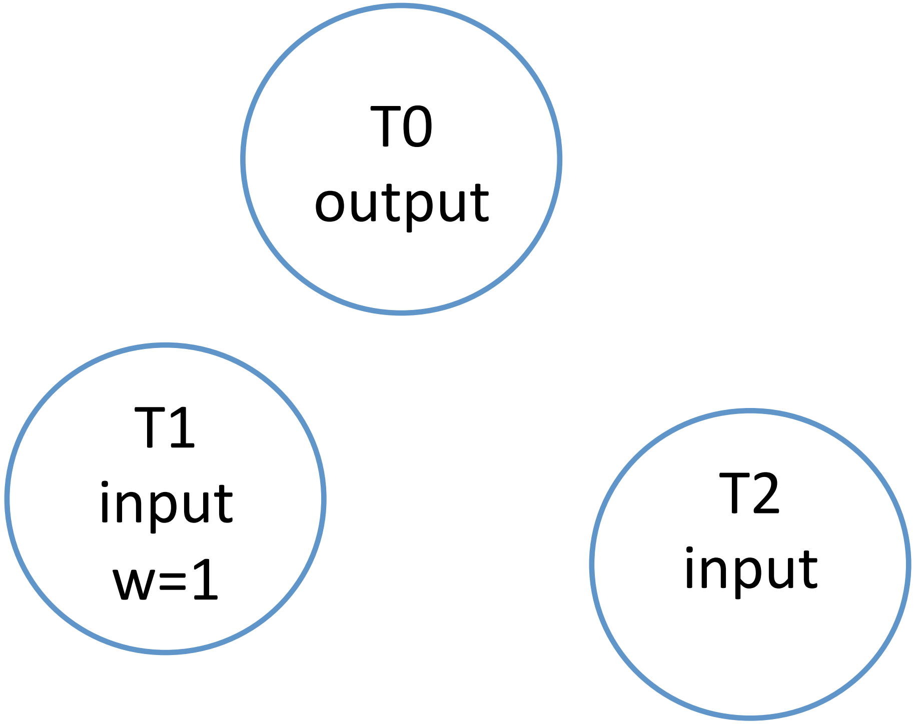
Ticket-Based Dependency Tracking



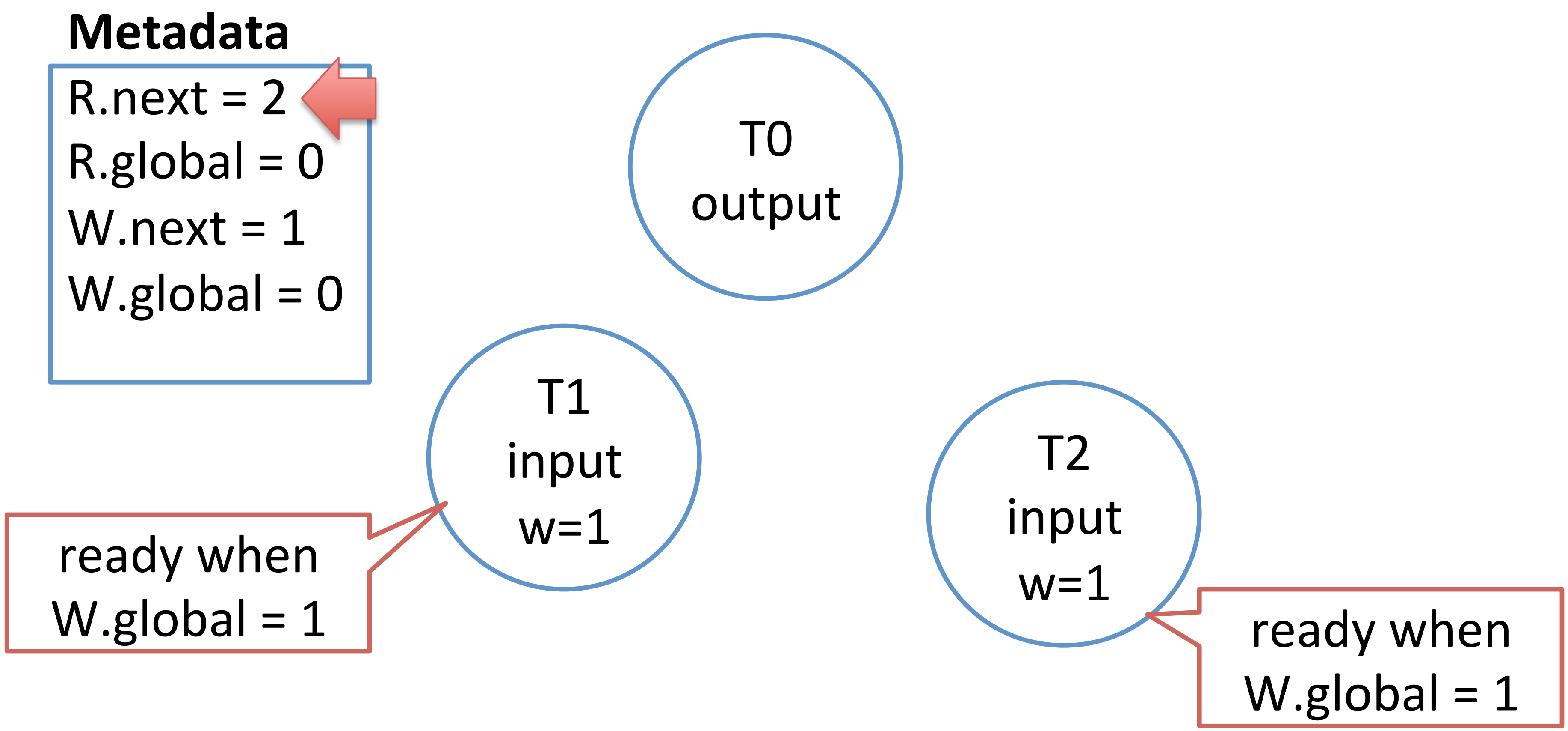
Ticket-Based Dependency Tracking

Metadata

R.next = 1
R.global = 0
W.next = 1
W.global = 0

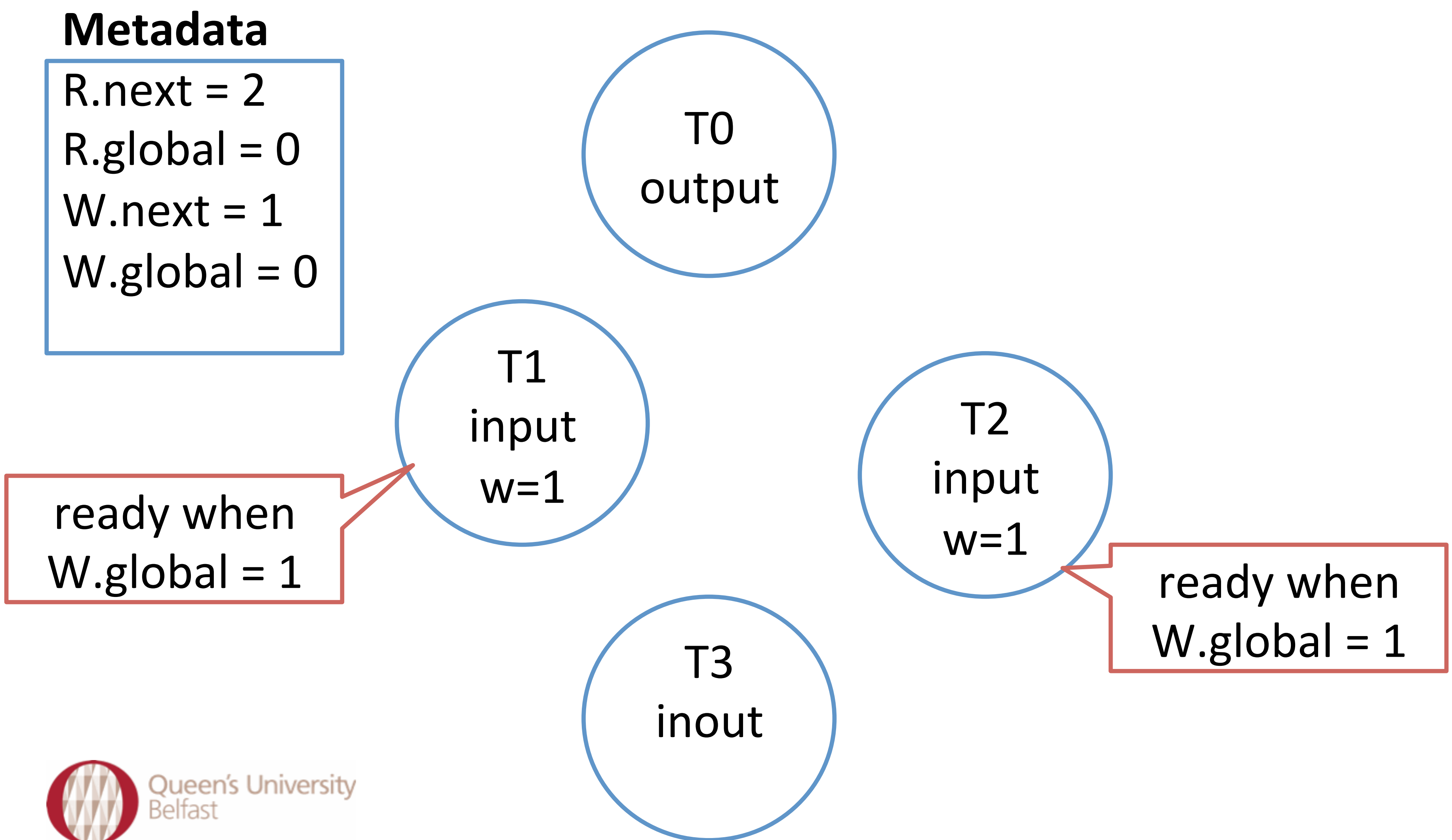


Ticket-Based Dependency Tracking

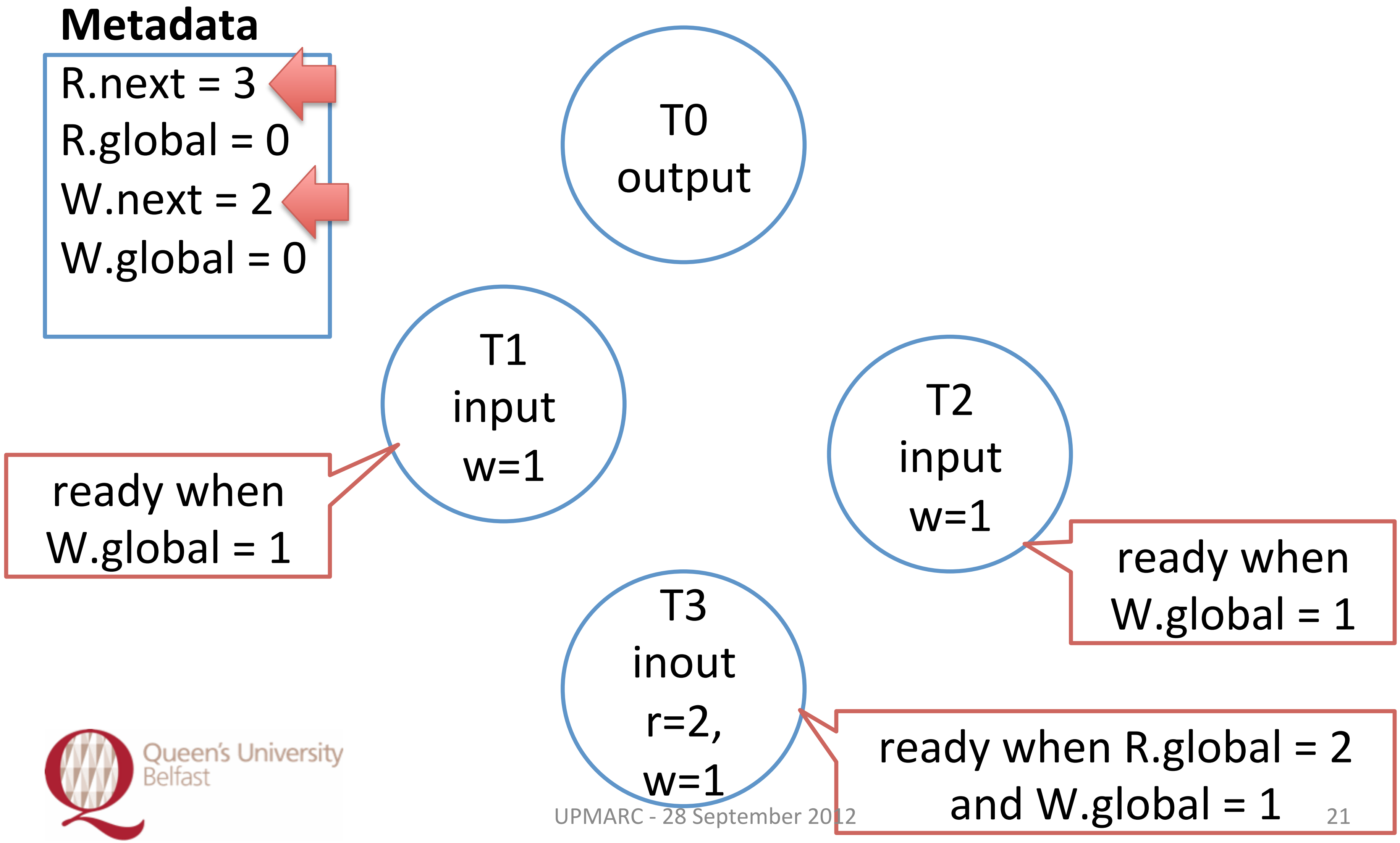


Queen's University Belfast

Ticket-Based Dependency Tracking



Ticket-Based Dependency Tracking



Dependency Tracking with Tickets

- Benefits
  - O(1) space overhead per task argument
  - O(1) time overhead per task argument
  - No locks; only atomic increments
  - Edge-based DAG:
    - Amortized O(1) space and time overhead for in, out, in/out
    - Constant depends on branching factor
- But: we don't store a list of ready tasks
  - Roots of the DAG
  - Because we do not have the edges
  - Judicious organization and traversal of pending tasks

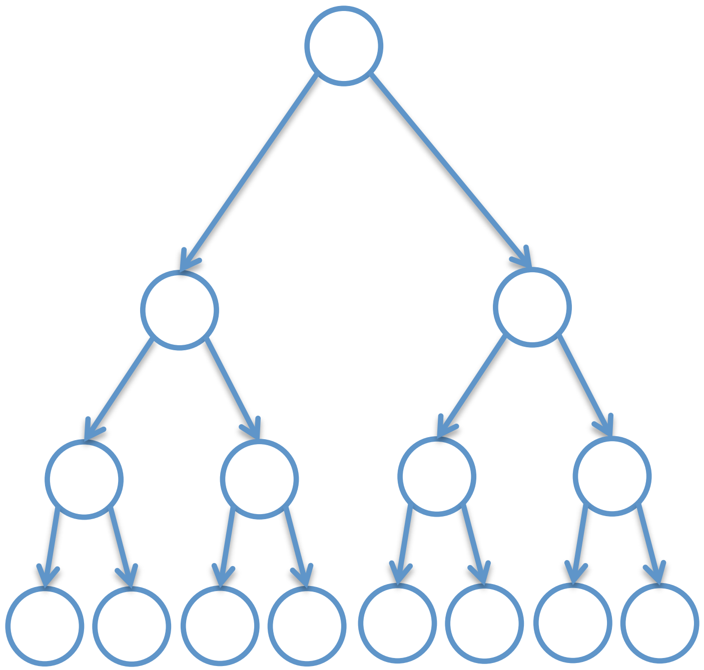
Contributions and Overview

- Efficient dependency tracking
  - versioned objects
  - with tickets, without edges in the DAG
- Extend Cilk scheduler
  - to unify recursive parallelism with dataflow parallelism
- Experimentally evaluate performance
  - on par with Cilk++, outperforms SMPSS

Queen's University Belfast

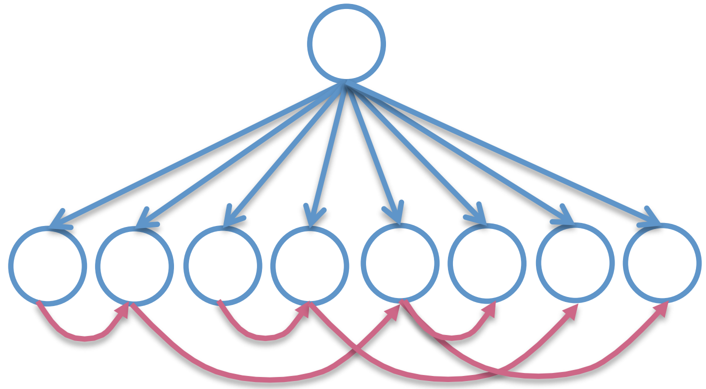
Unified Scheduler

Typical Cilk spawn tree



- Deep spawn tree

Typical task dataflow spawn tree

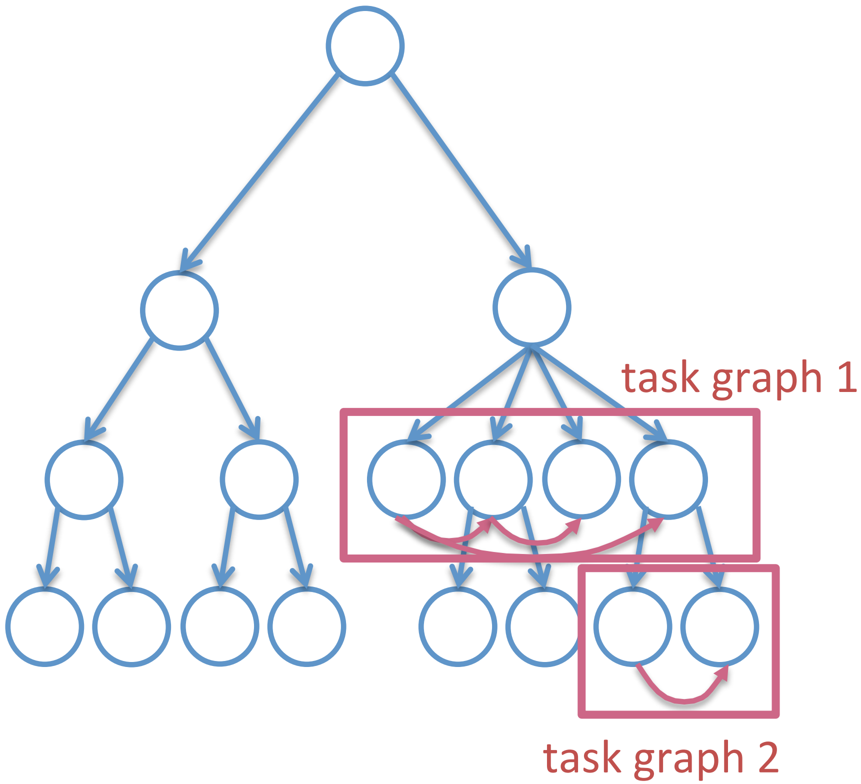


- Shallow spawn tree
- Dataflow dependencies between children
- Every task in the spawn tree may organize its children in a dataflow graph
- Arbitrary nesting of fork/join and task graphs

Queen's University Belfast

Unified Scheduler

Mixed fork/join – dataflow spawn tree



Qualitative properties

- Cannot maintain busy-leaves principle
  - Non-ready tasks are non-busy leaves
- Maintains work-first principle
  - Execute task immediately if data dependencies allow it
  - Keeps the task graph small
- Extend work-stealing rules
  - Take pending tasks into account
  - When returning from a procedure (provably-good-steal)
  - In random work stealing
- Stealing in dataflow graphs generally uses the expensive path

Queen's University Belfast

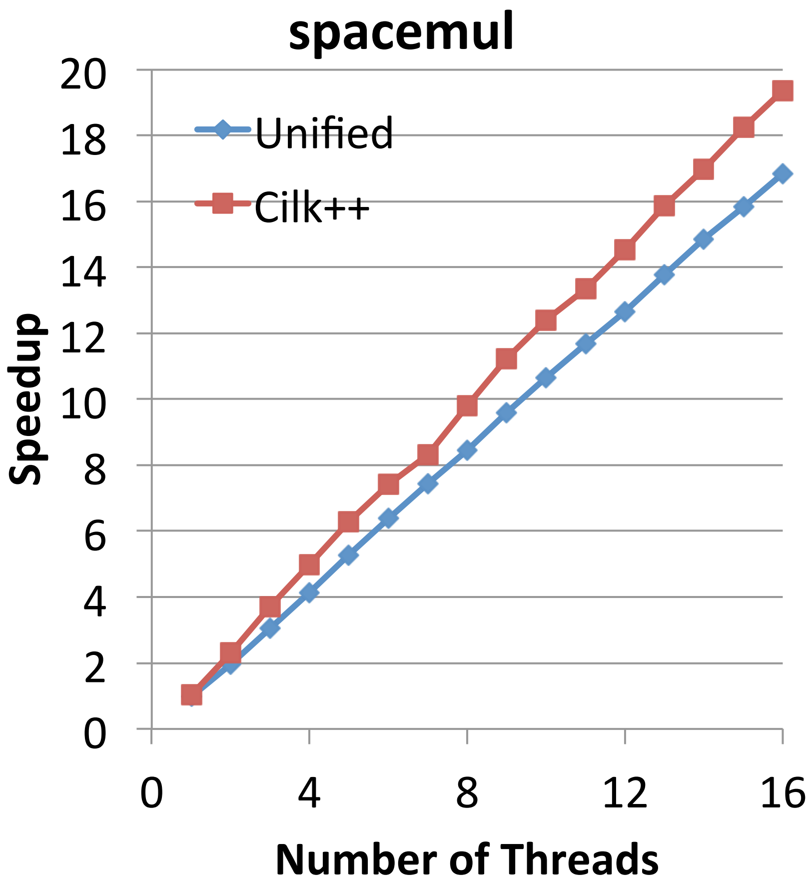
Contributions and Overview

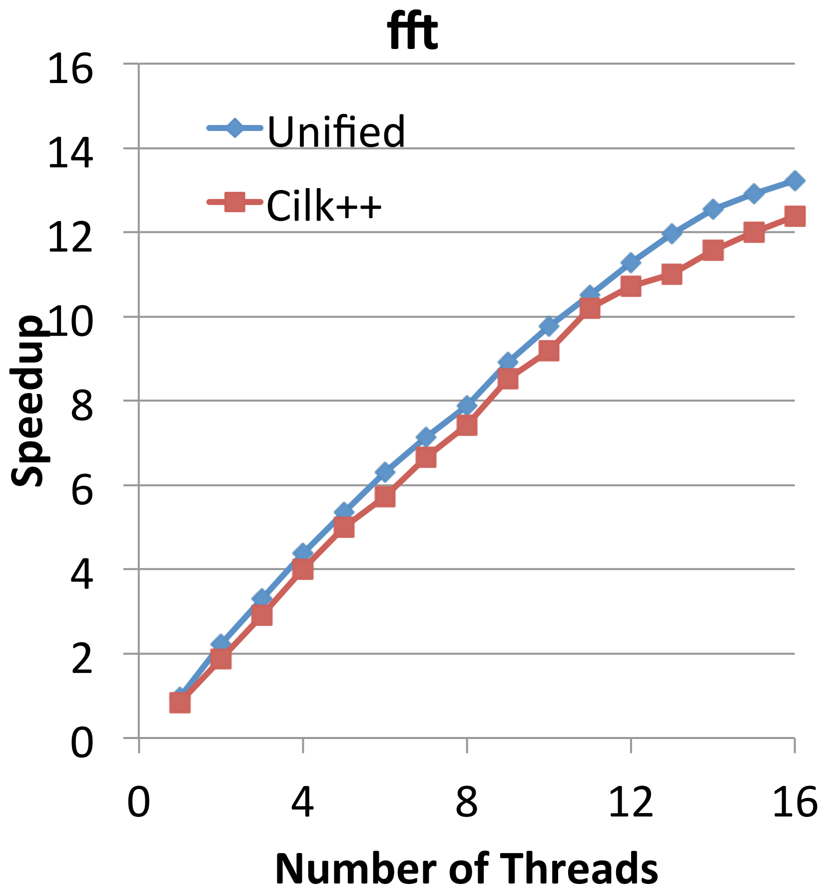
- Efficient dependency tracking
  - versioned objects
  - with tickets, without edges in the DAG
- Extend Cilk scheduler
  - to unify recursive parallelism with dataflow parallelism
- Experimentally evaluate performance
  - on par with Cilk++, outperforms SMPSS

Evaluation

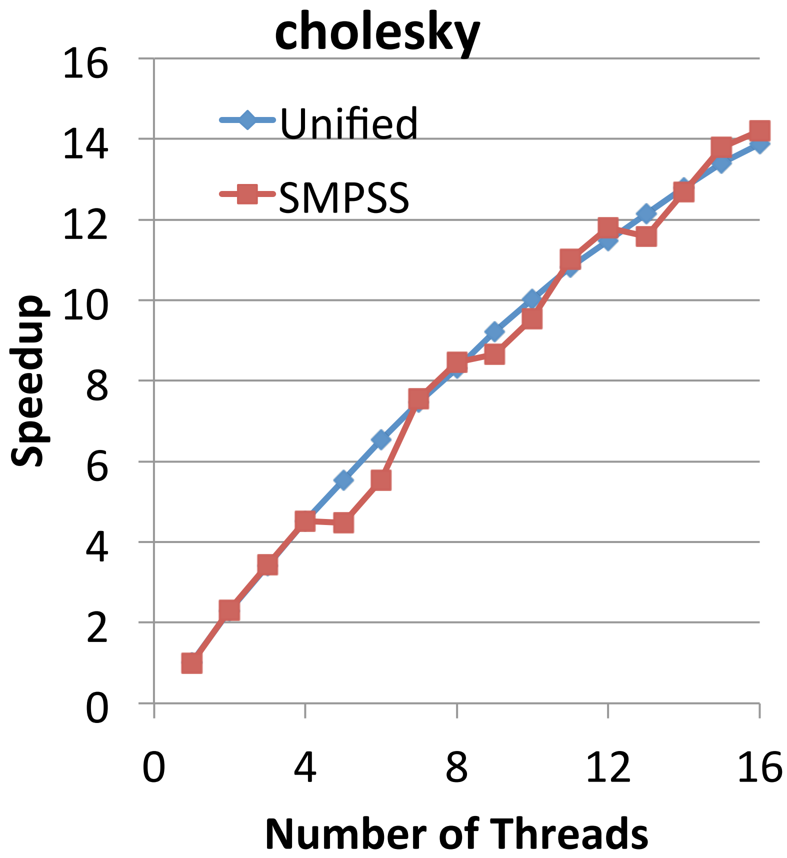
- Methodology
  - Implemented scheduler and language as C++0x library
  - Compare to Cilk++ on Cilk benchmarks
  - Compare to SMPSS on SMPSS benchmarks
- Platform
  - 4 quad-core Opteron 8350 HE @ 2GHz, 4 NUMA nodes
  - Ubuntu 9.10
  - Compilers:
    - Unified: gcc 4.6
    - Cilk++: gcc 4.2.4 extension
    - SMPSS v2.3: custom compiler (Mercurium + gcc)
  - Optimization level –O4
  - Some kernels use BLAS: GotoBLAS2, rev 1.13

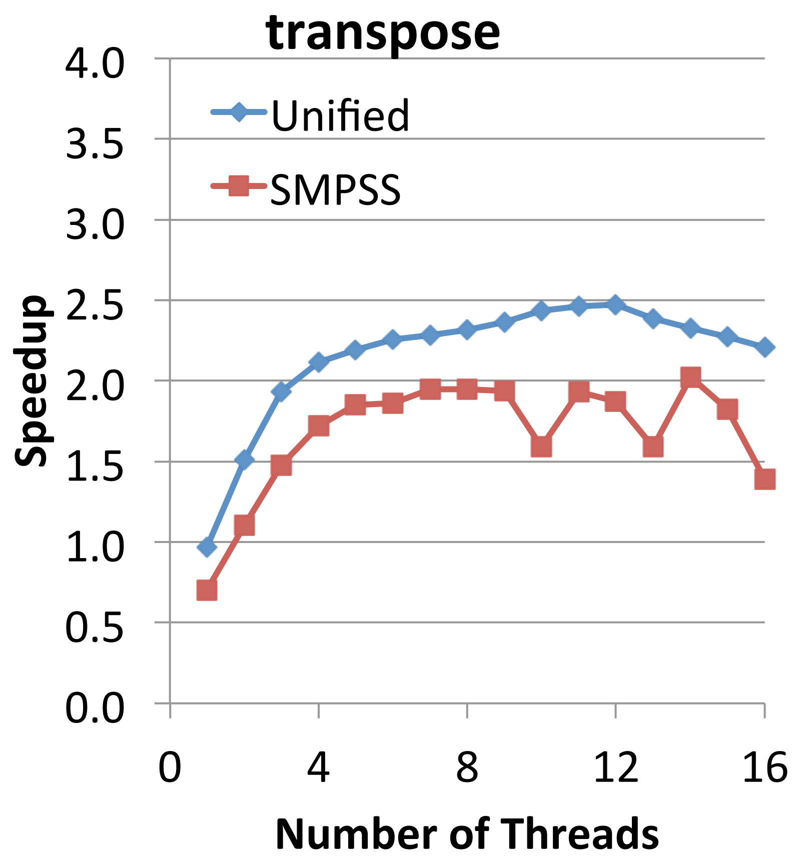
Comparison to Cilk++



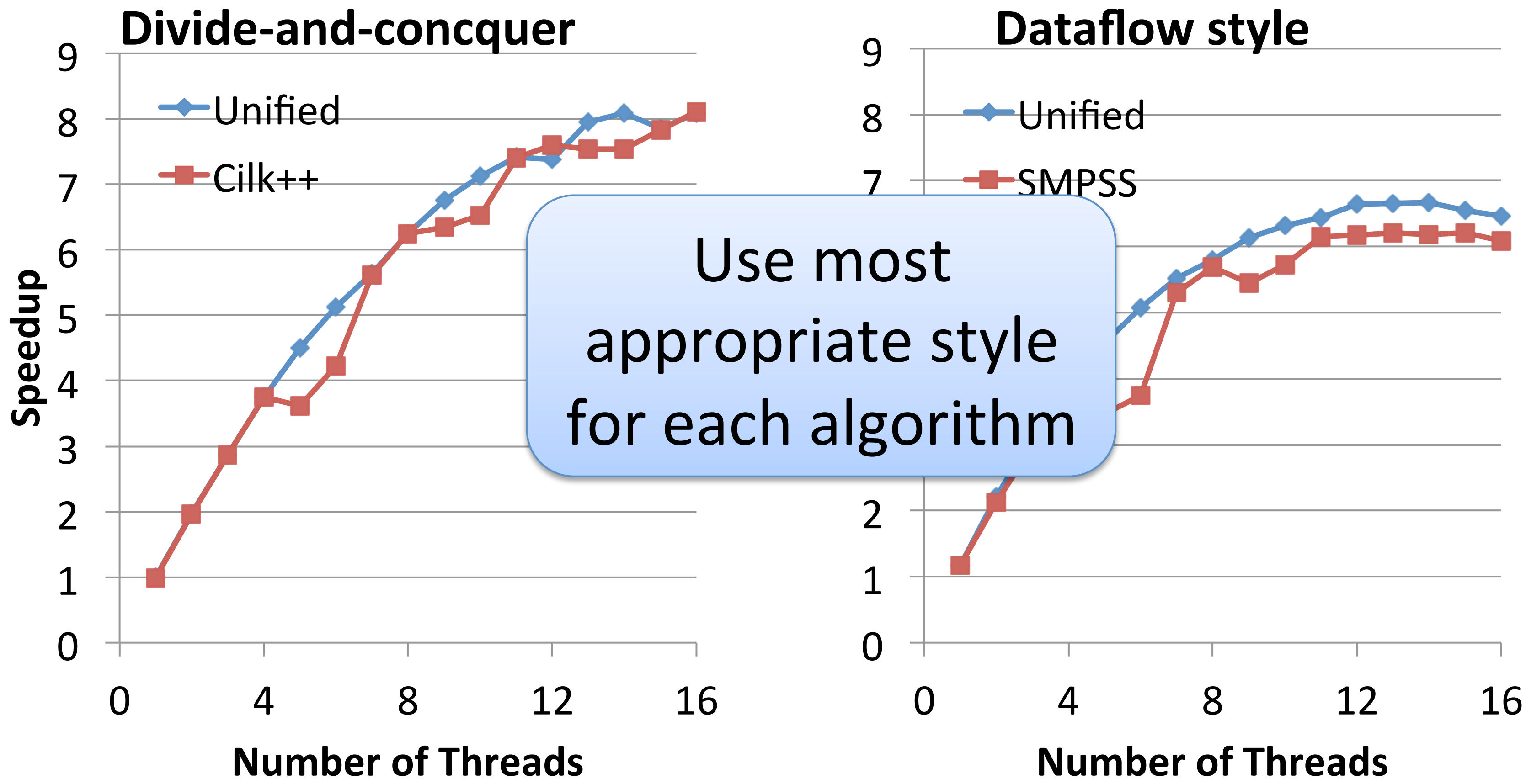


Comparison to SMPSS





Divide-and-conquer vs. Task Dataflow

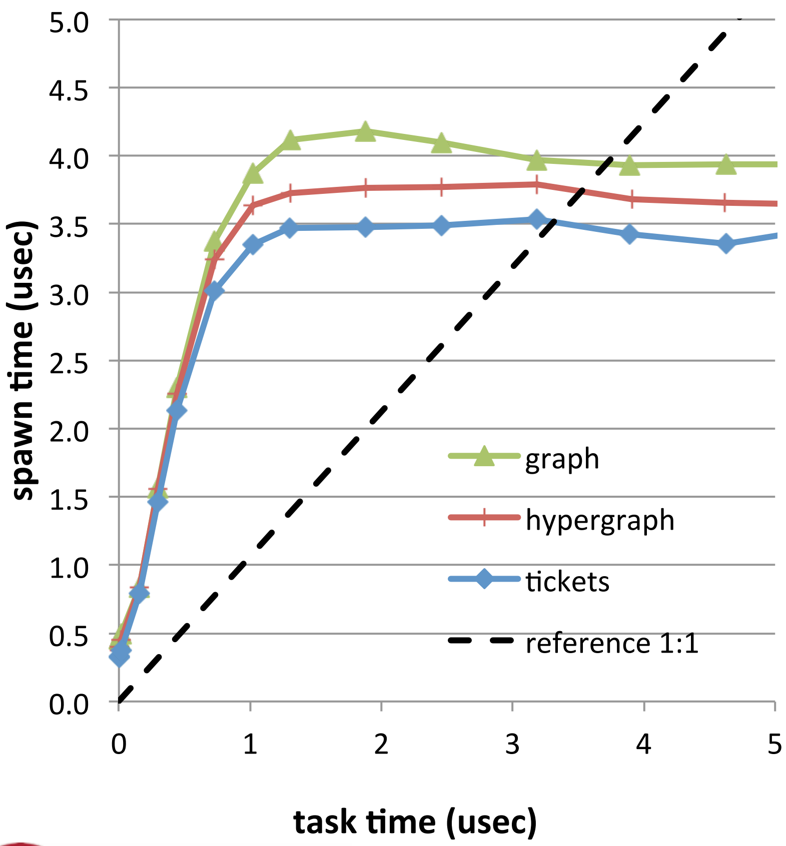


Use most appropriate style for each algorithm

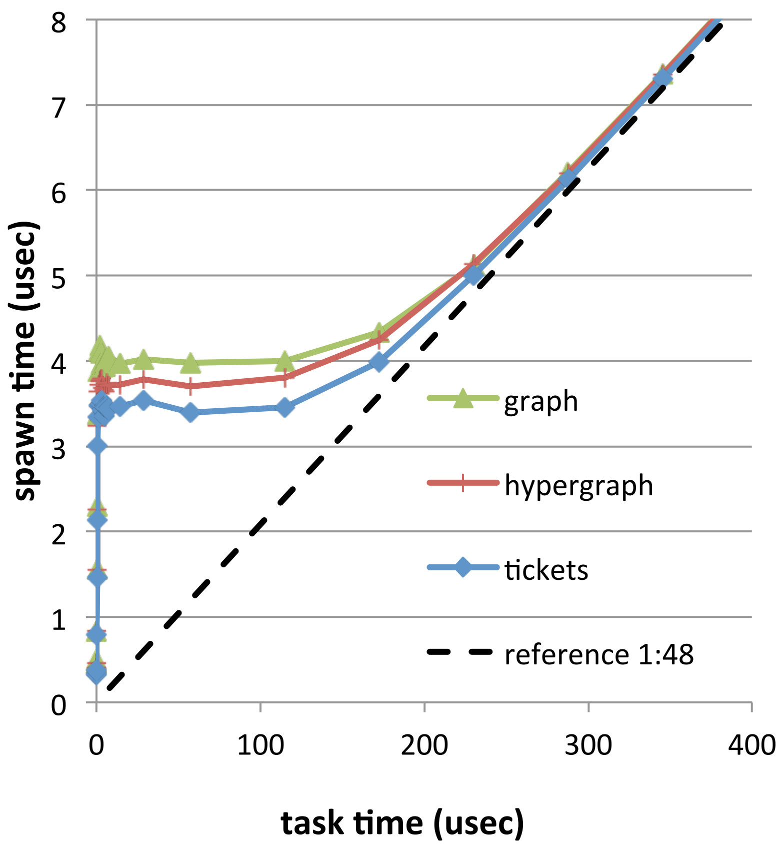
Matrix multiply, 4Kx4K matrix, 64x64 blocks

Tickets Make Most Difference When Scheduler is Stressed

Start of Speedup



Start of Linear Scaling



Queen's University Belfast

Opteron 6172 2.1 GHz, 8x hexa = 48 cores

Conclusion

- Task dataflow parallelism
  - Simple and widely applicable pattern of parallelism
  - Divide-and-conquer remains equally relevant!
- Unified scheduler
  - Adopts Cilk's work-first and work stealing principles
  - Extends single procedure body with dataflow scheduling
  - Versioned objects store metadata and simplify versioning
  - Efficient data dependency tracking with tickets
- Evaluation demonstrates performance
  - On par with Cilk++
  - Outperforms SMPSS, a task dataflow-aware scheduler

Thank You!

http://www.github.com/hvdieren/swan