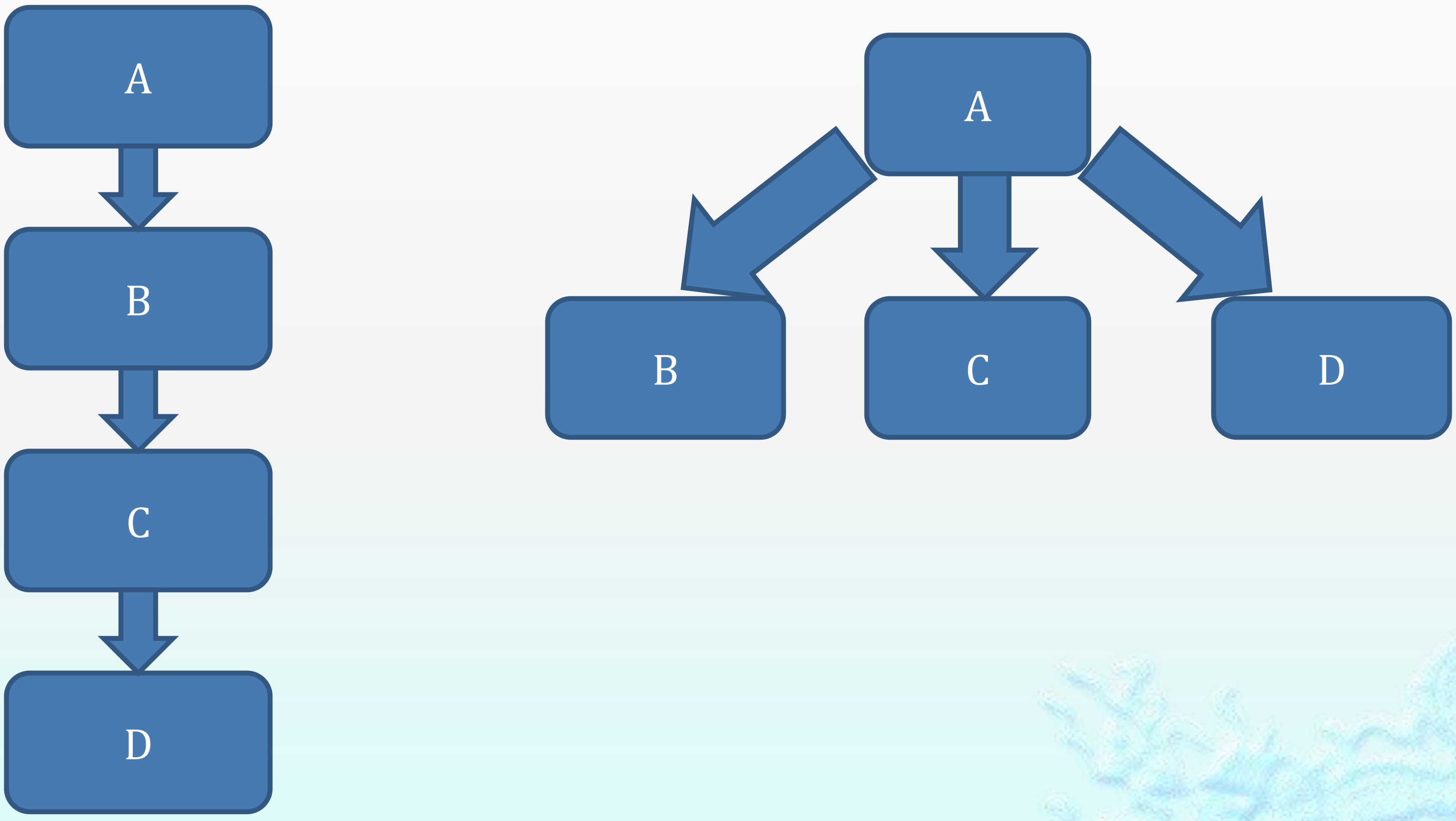
Automatic Parallelism Discovery

Hongyu Gao

Introduction



Sequential vs Parallel execution

Introduction

- Why do we need parallel execution?
  - Ever increasing computation scale
  - Limited computational power of a single processor
  - Large scale computation infrastructure available
    - IBM Blue Gene/P, 1PFLOPS with 294,912 processors

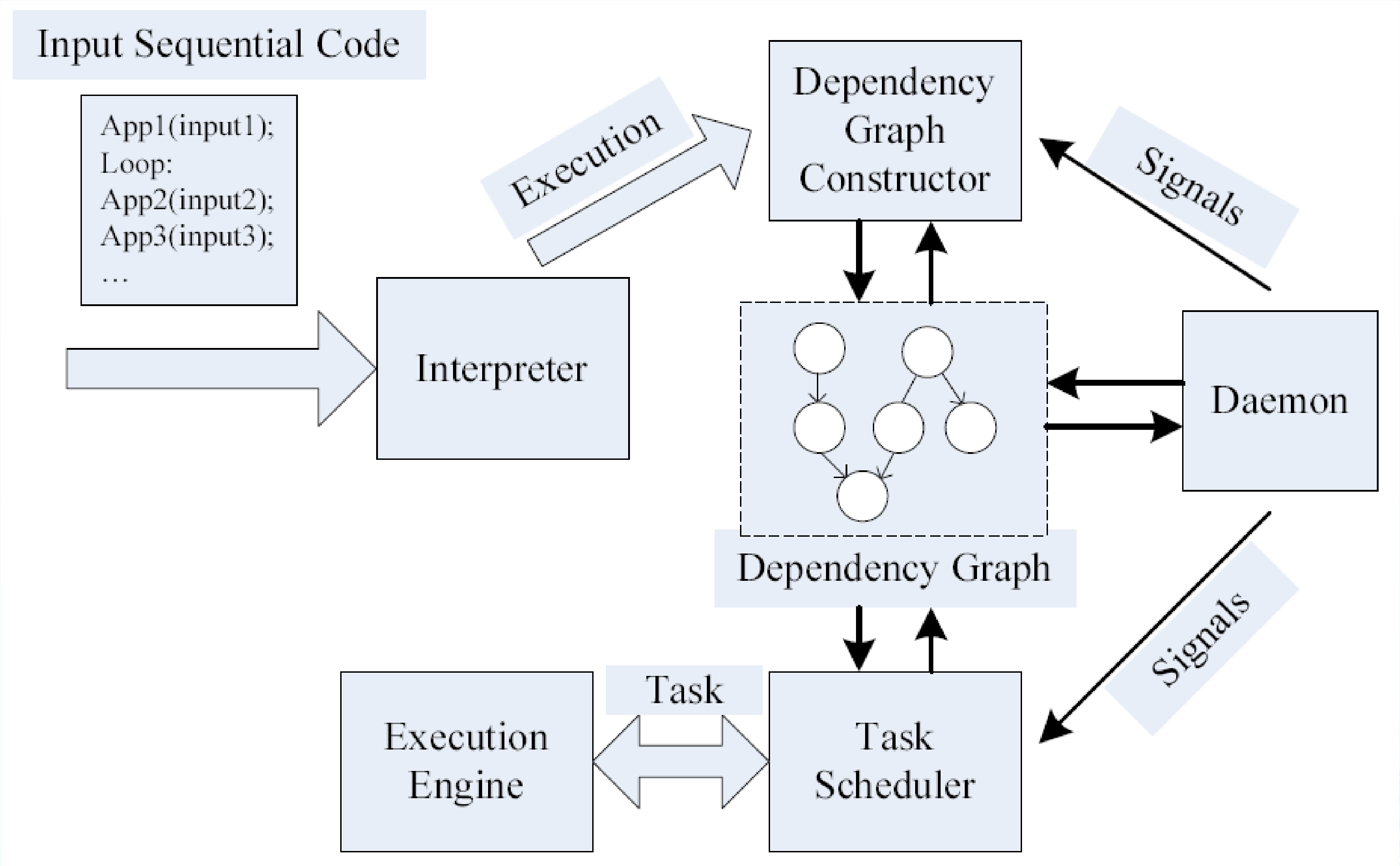
Introduction

- A dilemma:
  - Emerging need for parallel computing
  - Difficulty of parallel programming
- A solution:
  - Automatic parallelization of sequential program

Proposed Solution

- A system that
  - Takes in sequential program
  - Automatically reveals potential parallelism
  - Automatically executes the program in parallel

Proposed Solution, cont'd



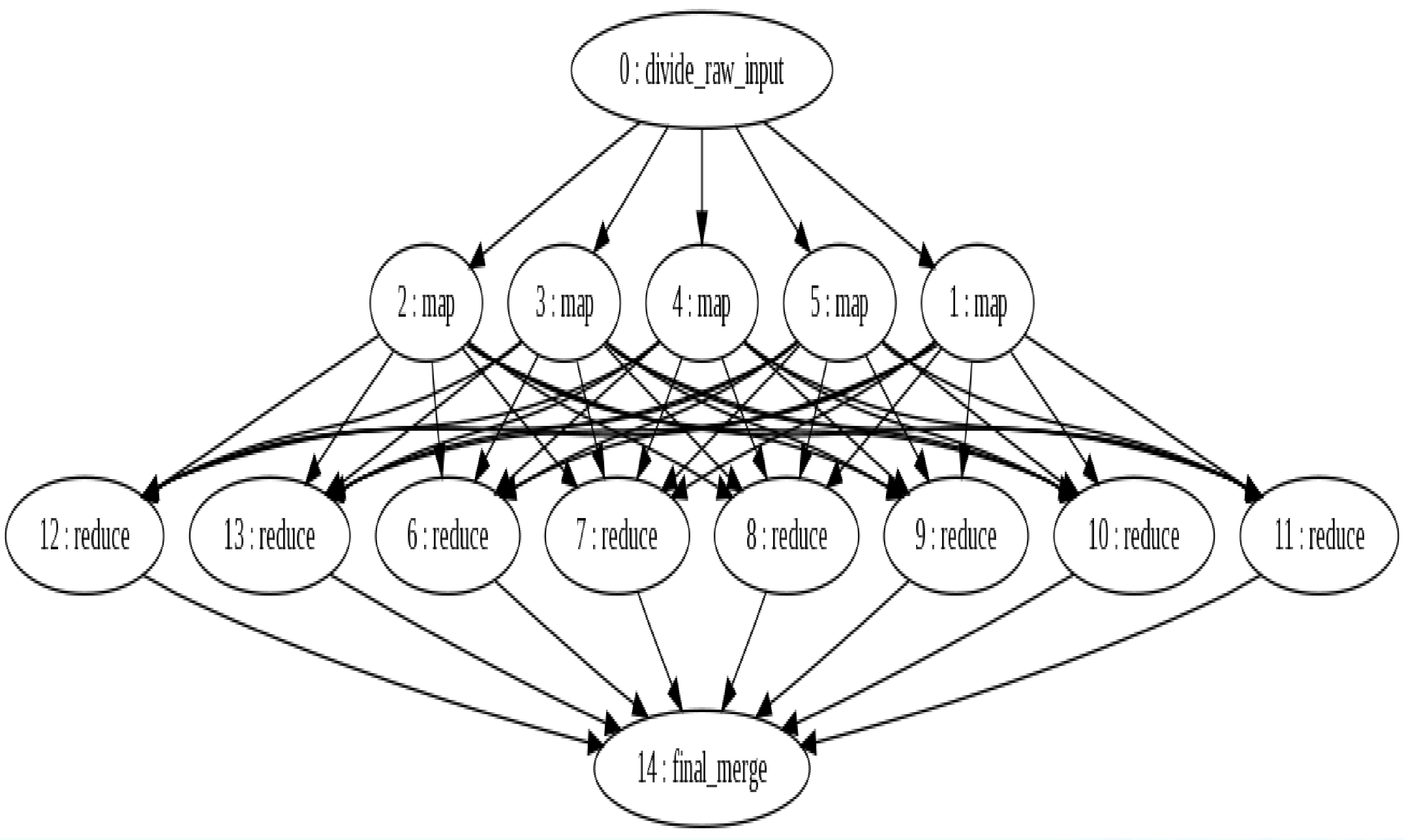
Dependency graph generation

- A directed acyclic graph
- A node:

  The smallest block of code that is scheduled for parallel execution
- An edge:

  A node depends on the completion of another node before it can be executed

An example



0 : divide_raw_input
2 : map
3 : map
4 : map
5 : map
1 : map
12 : reduce
13 : reduce
6 : reduce
7 : reduce
8 : reduce
9 : reduce
10 : reduce
11 : reduce
14 : final_merge

Dependency graph constructor

Algorithm 1 InsertNewNode()

$n \leftarrow thenewnode$
$G \leftarrow thecurrentdependencygraph$
Foreach $v \in G.nodes$
  If $(v.output \cap n.input \neq \emptyset$
    $or\ v.output \cap n.output \neq \emptyset)$
    $n.counter++$
    $n.depend.insert(v)$
    $v.be\_depended.insert(n)$
  EndIf
EndForeach
$G.nodes.insert(n)$
If $(n.depend = \emptyset)$
  $n.type \leftarrow ready$
  $G.ready\_nodes.insert(n)$
Else
  $n.type \leftarrow blocking$
EndIf

Task scheduler

- A node (task) can be scheduled if:
  - It has no in-edge
  - All nodes that it depends on have been completed

Task scheduler

Algorithm 2 TaskCompletion()

$wait(sig\_task\_complete)$
$t \leftarrow thetaskthatjustcompletes$
$G \leftarrow thecurrentdependencygraph$
$t.type \leftarrow done$
Foreach $n \in t.be\_depended$
    $n.counter - -$
    $n.depend.remove(t)$
    If $(n.counter = 0)$
        $n.type \leftarrow ready$
        $G.ready\_nodes.insert(n)$
        $signal(sig\_node\_ready)$
    EndIf
EndForeach

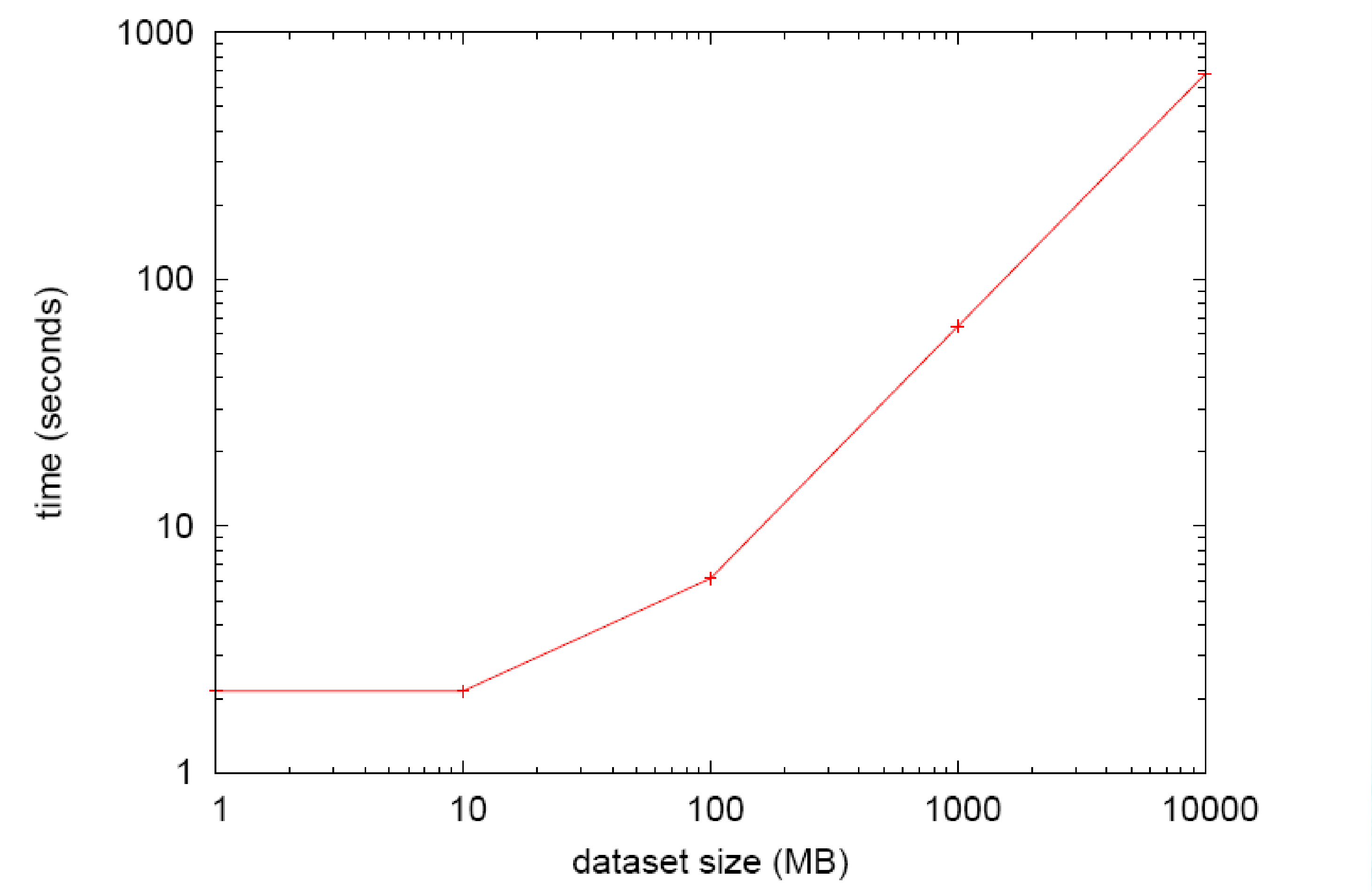
Pipelined execution

- Observation 1: The *ready* node in the dependency graph can be scheduled even before the graph is completely built.
- Observation 2: All the undeterministic factors that prevents the construction of the complete dependency graph can be resolved by executing the partial graph that has been constructed

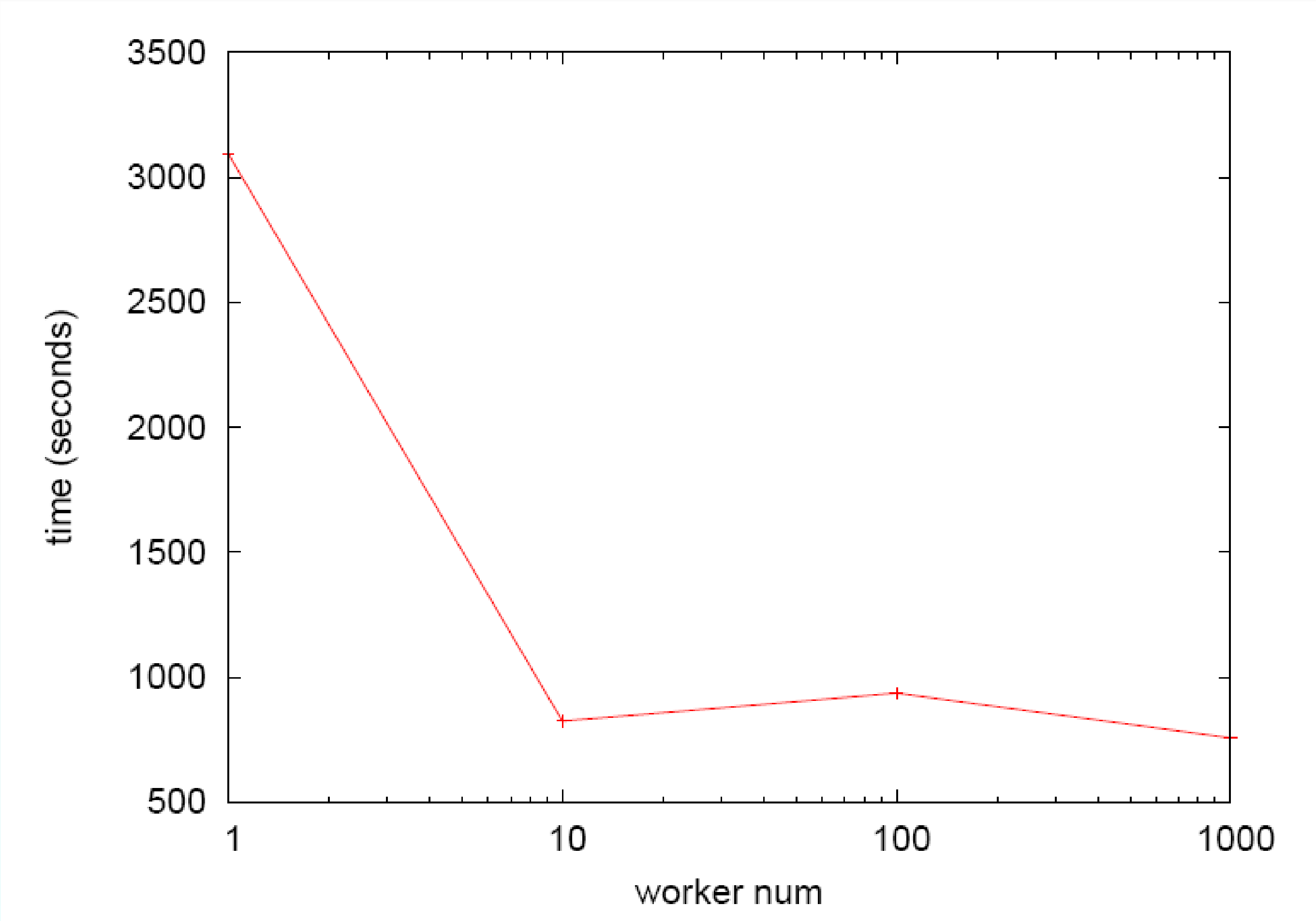
Pipelined execution

- Multiprocess design:
  - graph constructor incrementally inserts new nodes into the graph.
    - A window size limitation
  - Task scheduler blocks waiting for either a node is ready or an execution has completed

Experimental results



Experimental results, cont'd



Questions?

Thank you!