

Scheduling for Sensor Networks

Maria Karlsson
Welf Löwe



Lois Space Centre and Växjö University

Outline

- **Basic scheduling:** beam-forming and beam-broadcasting programs
- **Local scheduling:** one program
- **Global scheduling:** set of programs
- **Competitive scheduling:** set of competing programs

(Scientific) Program Design

Exact Solution

$$\delta'' - \alpha^2(\delta - \delta_A) = 0$$

$$\delta(0) = \delta_0 \quad \delta(L) = \delta_L$$

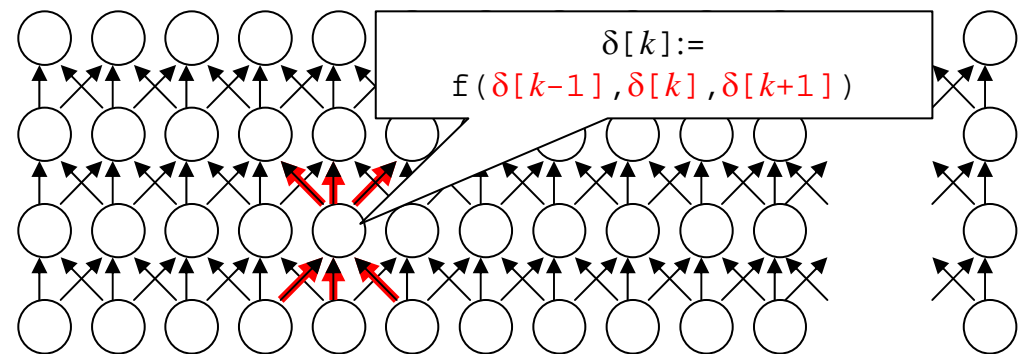
Numeric Approach

$$\delta^{(t+1)}(x) = (1 - \omega) \delta^{(t)}(x_k) + \omega(-2/\Delta^2 - \alpha^2)^{-1} \times (-\alpha^2 \delta_A - \delta^{(t)}(x_{k-1})/\Delta^2 - \delta^{(t)}(x_{k+1})/\Delta^2)$$

Data-parallel program

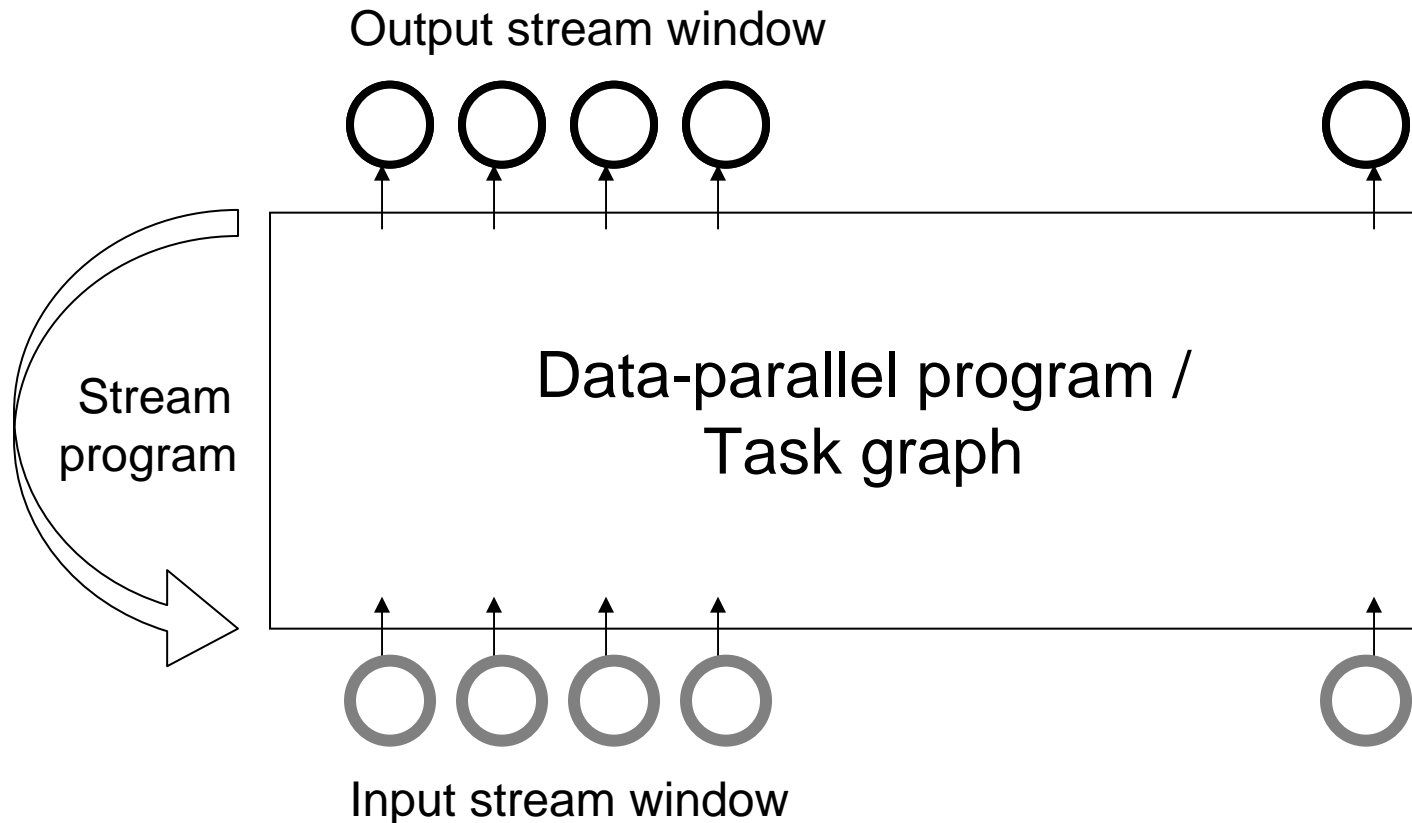
```
for t := 1..s do
  forall k := 1..n-2 pardo
     $\delta[k] := f(\delta[k-1], \delta[k], \delta[k+1])$ 
```

Task graph



Scheduling: maps all tasks t to $(processors, time)_t$

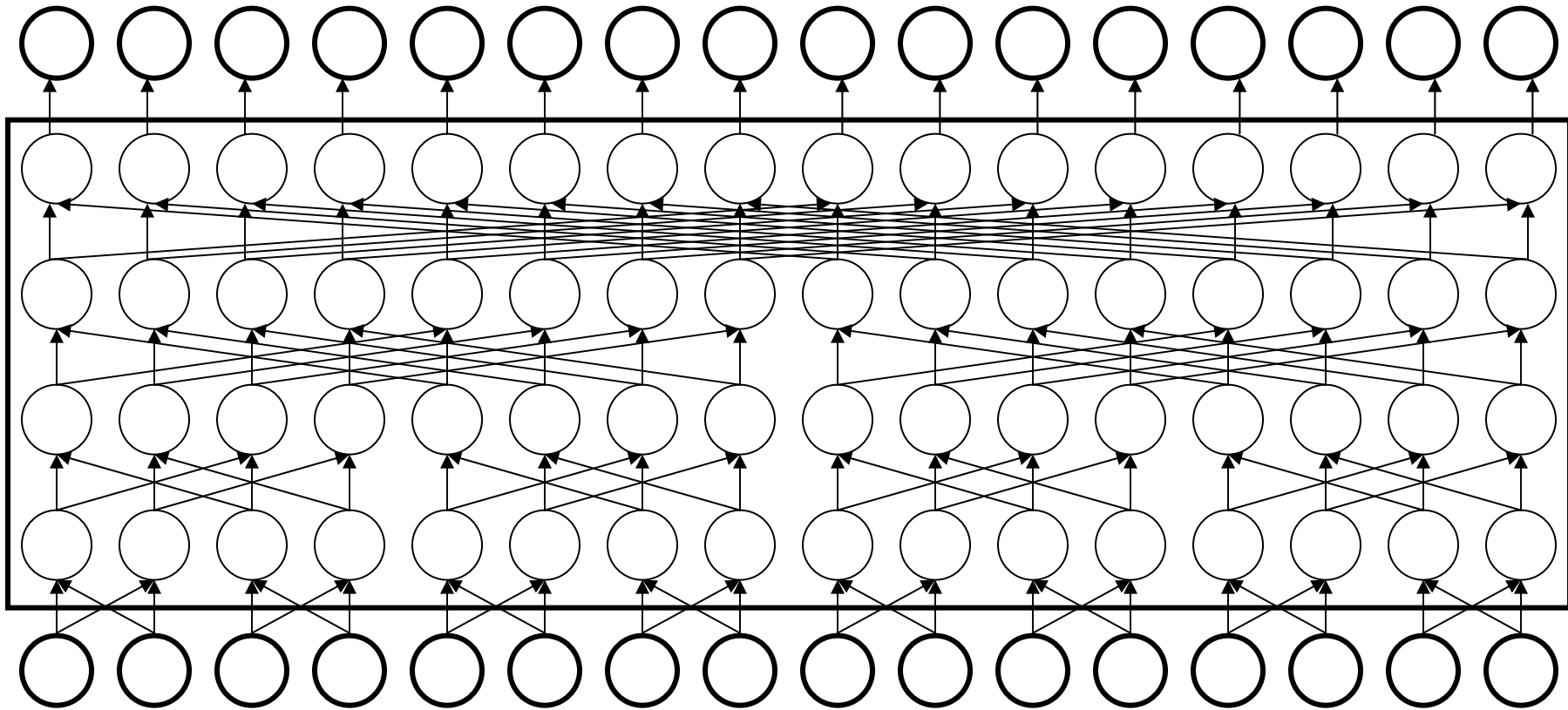
Stream Programs



Goal

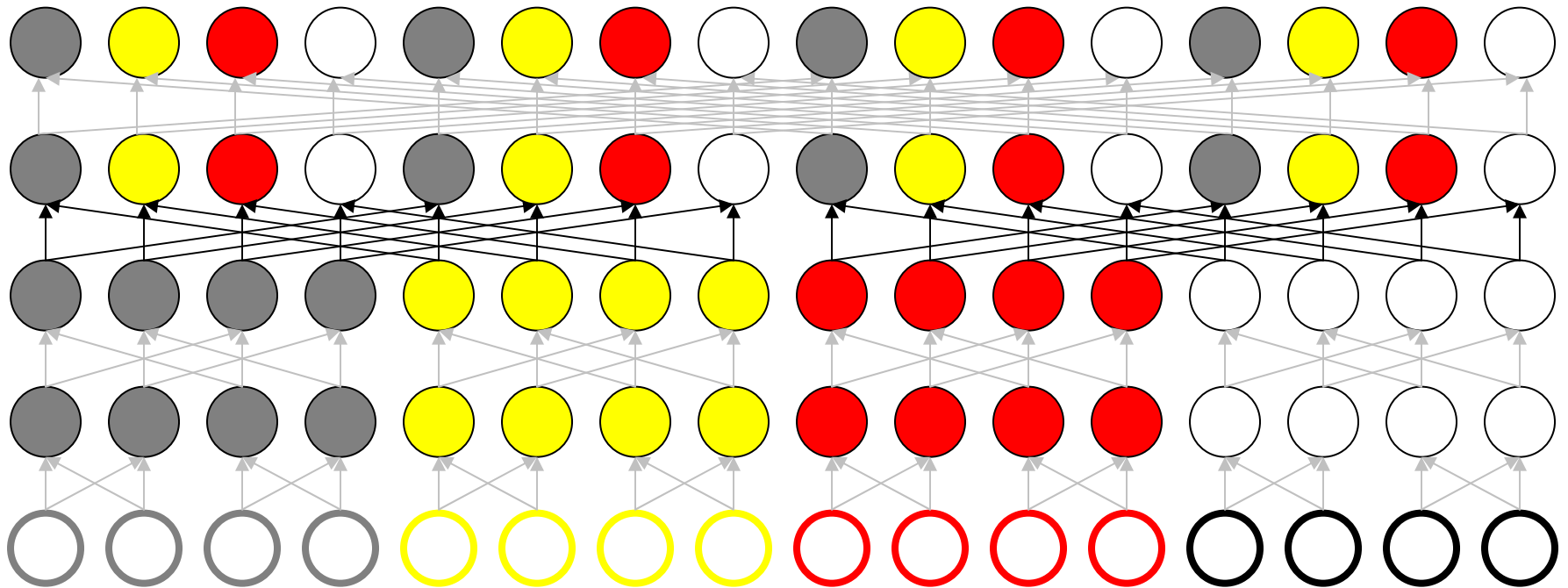
- Reducing completion time for schedule of a task graph increases throughput
- Completion time called makespan for a schedule:
 - Completion time for a task t : $time_t + c(t)$
 - **Makspan** of a schedule is the largest completion time of a task
- Scheduling is an optimization
 - Compute a correct schedule minimizing its makespan
 - **NP-hard**, not optimally solvable in acceptable time
 - Need approximations

Example: FFT



Input stream window size = Output stream window size = 16

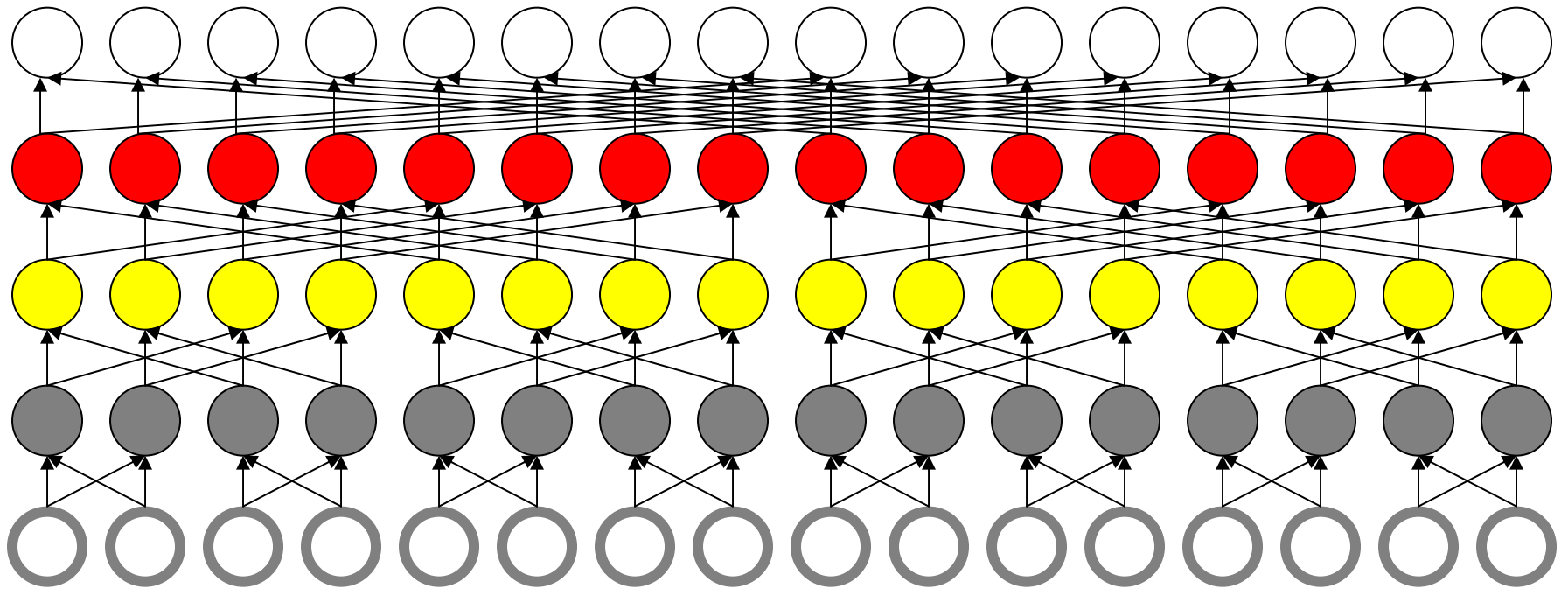
Schedule: FFT on 4 processors



Completion time: $16 \times \text{comp} + 3 \times \text{send}(1) + 3 \times \text{recv}(1) + \text{Latency}$

Throughput: $1 / (16 \times \text{comp} + 3 \times \text{send}(1) + 3 \times \text{recv}(1) + \text{Latency})$

Schedule: FFT (alternative)



Completion time: $4 \times (16 \times \text{comp} + \text{send}(32) + \text{recv}(32) + \text{Latency})$

Throughput : $1 / (16 \times \text{comp} + \text{send}(32) + \text{recv}(32))$

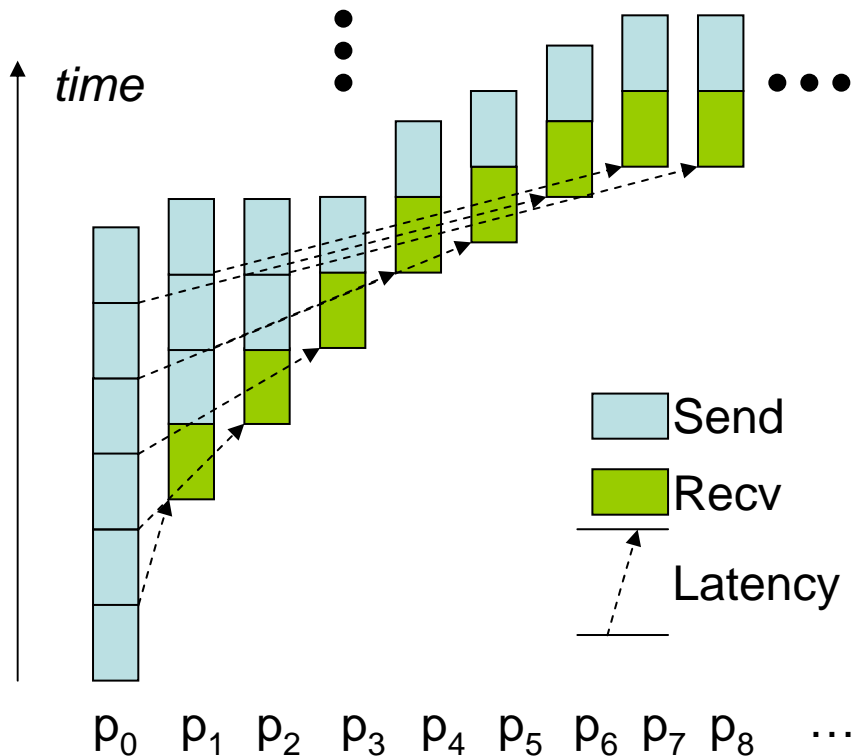
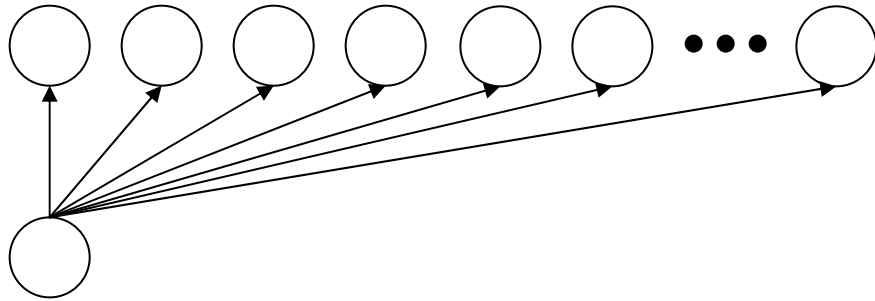
Outline

- **Basic scheduling:** beam-forming and beam-broadcasting program
- Local scheduling: one program
- Global scheduling: set of programs
- Competitive scheduling: set of competing programs

Approach

- Programs
 - Reduction, broadcast/multicast trees
 - Family of programs: statically known but need adapt to
 - Configurations by user
 - Hardware changes adding/removing/failure of sensors, processors, network links
 - Automated (re-)scheduling has advantages
- Scheduling technique
 - Scheduling schema for reduction, broadcast/multicast tree families
 - Configurable
 - (almost) optimal
 - Defines a family of schedules
 - Select and (re-)deploy the appropriate (new) schedule instance

Example: Simple broadcast



- Family of task graphs: broadcast to P processor
- Broadcast tree [Karp et al.] defines family of optimal schedules
- Changing # processor P
 - Select sub-schedule with min makespan that reaches P processors
- Changing network
 - Send, Recv, Latency changes
 - New broadcast tree

Outline

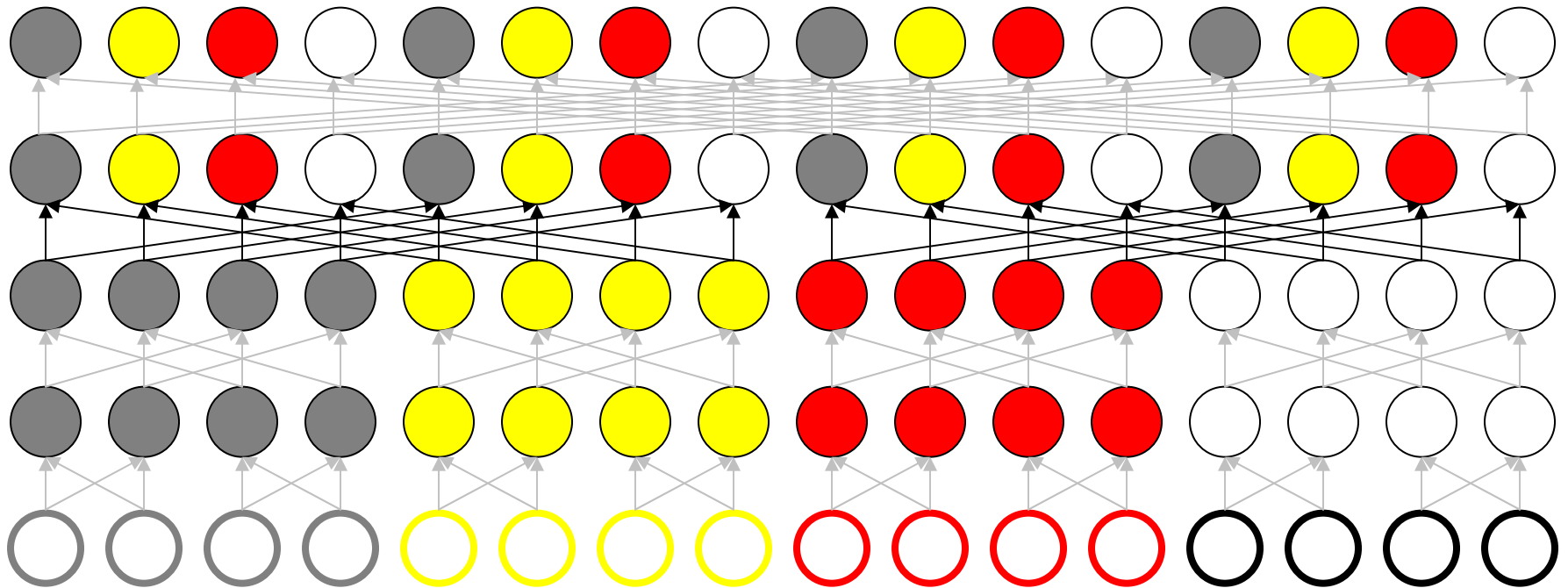
- Basic scheduling: beam-forming and beam-broadcasting programs
- **Local scheduling: one application**
- Global scheduling: set of programs
- Competitive scheduling: set if competing programs

Approach

- Program
 - Not known in advance,
 - If oblivious (most numeric problems lead to oblivious programs) task graphs can be derived automatically,
 - Contain calls to other (oblivious) programs
 - Scheduling technique
 - Task graph scheduling and malleable task scheduling
 - Malleable task graph scheduling
- approximate optimum solution

Task Graph Scheduling

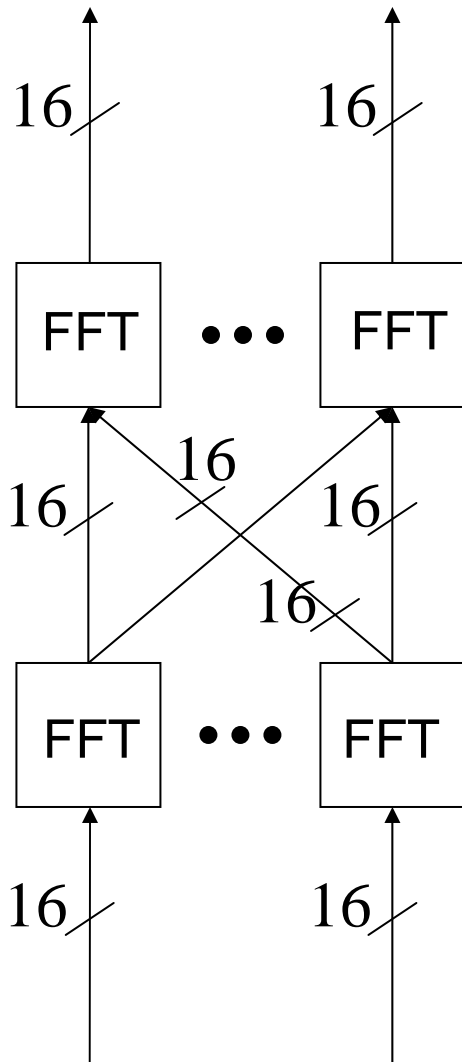
FFT on 4 processors



- Computational tasks, each computable on one processor
- Approximations to min makespan scheduling exist

Malleable task:

Example: 2D FFT



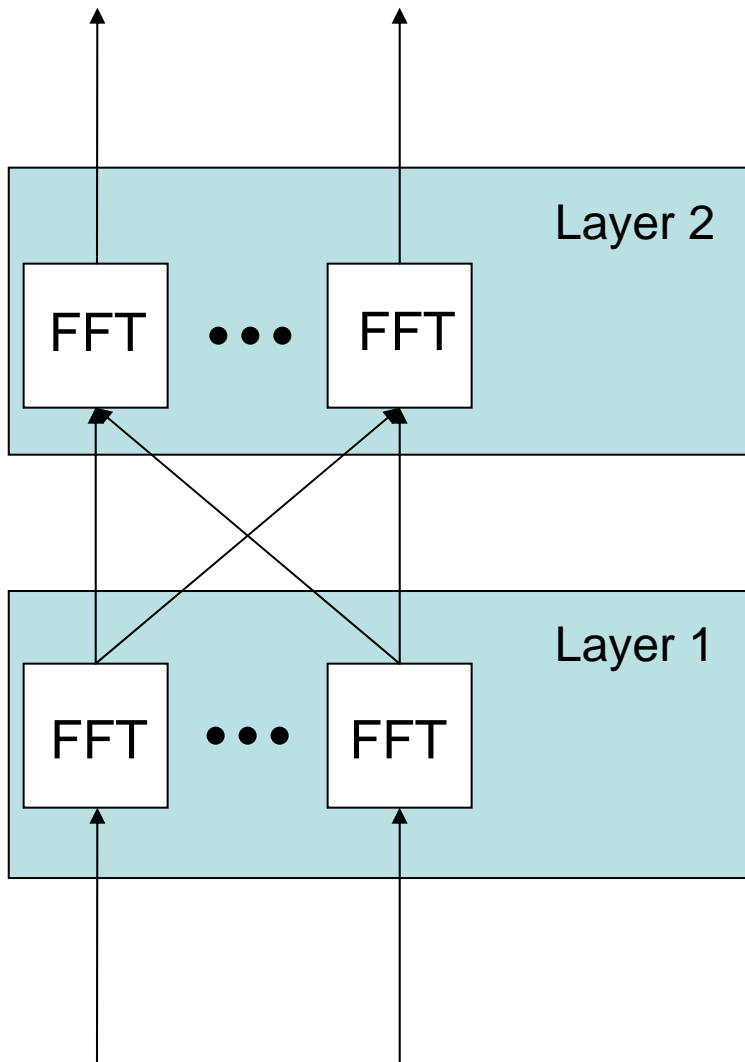
- Task graph contains
 - Simple tasks (operations, assignments, etc.)
 - Calls to task graphs
 - Recursive
- Malleable Tasks:
 - Can be executed on more than one processor
 - Non-increasing completion time in p
- **Idea:** Each procedure call corresponds to a malleable task
- Approximations to min makespan scheduling exist for independent malleable tasks

Task Graph \bowtie Malleable Task

- Given a task graph tg
- For all processor $p \in 1..P_{max}$
 - Compute $schedule(tg, p)$
using task graph scheduling
 - Set $completion(tg, p) =$
 $makespan(schedule(tg, p))$
 - If $completion(tg, p) > completion(tg, p-1)$
 - Set $completion(tg, p) = completion(tg, p-1)$
(guarantees non-increasing completion times)

Malleable Task Graph Scheduling

Example: 2D FFT



- Compute malleable tasks for **task graphs** behind each procedure call
- For each layer:
 - Schedule the independent malleable tasks (tasks of a layer are independent by definition)
 - Schedule a multicast (all-to-all) communication
- Continues recursively for calls to malleable task graphs,
- Recursively, compute malleable tasks for **malleable task graphs**

Malleable Task Graph \Leftrightarrow Malleable Task

- Given a **malleable** task graph tg
- For all processor $p \in 1..P_{max}$
 - Compute $schedule(tg, p)$
using **malleable** task graph scheduling
 - Set $completion(tg, p) = makespan(schedule(tg, p))$
 - If $completion(tg, p) > completion(tg, p-1)$
 - Set $completion(tg, p) = completion(tg, p-1)$
(guarantees non-increasing completion times)

Outline

- Basic scheduling: beam-forming and beam-broadcasting programs
- Local scheduling: one program
- **Global scheduling: set of programs**
- Competitive scheduling: set if competing programs

Approach

- Programs
 - Set of independent programs
 - Assumed to be oblivious, hence, each
 - Correspond to a (malleable) task graph
 - Schedulable using (malleable) task graph scheduling
- Scheduling technique trivial
 - Assume a virtual main-program calling programs independently hence, in parallel
 - Just another recursion level
 - Apply independent malleable task scheduling

Outline

- Basic scheduling: beam-forming and beam-broadcasting programs
- Local scheduling: one program
- Global scheduling: set of programs
- **Competitive scheduling: set of competing programs**

Approach

- Programs
 - Different research groups
 - Competing for limited resource: sensor-network
 - Campaign deployment
 - Selecting the most promising programs
 - Currently “Research council” approach
- Scheduling technique
 - Idea: Use market oriented approach, market based scheduling
 - Challenge: Find market rules such that egoistic behavior of groups is good for community
 - Advantages:
 - Automation
 - Decisions are fair and transparent
 - Better resource usage

Market rules and parameters

- Utility:
 - for a group to run a campaign
 - for the “owner” to rent out the sensor-network
- Costs:
 - for a group to charter the sensor-network
 - for the “owner” to maintain the sensor-network
- Currency \$ (Money)
- Assumptions on the behavior of market participants, e.g., egoism but truthfulness

Agents for Market and Participant

- Task agents (buyer):
 - Own a program modeled by a task graph, tg
 - Local schedules, $tg, p \rightsquigarrow ls, makespan$
 - Utility (income): $makespan \rightsquigarrow \$$
 - Bid: $(\$, makespan, ls)$
 - Goal: maximize surplus: utility - price
- Process agents (seller):
 - Own a resource, $p \rightsquigarrow [1 \dots P_{max}]$
 - Allocation: $\rightsquigarrow time$
 - Utility (costs): $time \rightsquigarrow \$$
 - Bid: $(\$, time)$
 - Goal: maximize surplus: price - utility
- Market:
 - Maps buy- to sell-bids
 - Sets the prices per time slot $(\$_1 \$_2 \dots \$_{Tmax})$
 - Goal: maximizes surplus: $\sum_{Buy-Bid} - \sum_{Sell-Bid}$

Market Process

1. For all processor agents pa :
 - Send current allocation and prices
 - Get new sell bids ($\$_{pa}, time_{pa}$)
2. For all tasks agents ta :
 - Send current allocation and prices
 - Get buy bit ($\$_{ta}, makespan_t, ls_{ta}$)
3. Terminate if no new bids.
4. Calculate prices per time slot ($\$_1 \$_2 \dots \$_{Tmax}$)
5. Calculate processor allocation, i.e., **globally schedule** gs_t maximizing $\sum_{for\ all\ ta} \$_{ta} - \sum_{for\ all\ pa} \$_{pa}$
6. Terminate if no improvements (after n iterations).
7. Go to 1.

Task Agent ta Process

1. Compute **local schedules**, $tg, p \in Is_p, makespan_p$
2. Get current processor prices and own current allocation
3. Compute surplus_{current allocation}
4. For all processor $p \in 1.. P_{max}$ and all times $t \in 0.. T_{max}$
:
 - Compute surplus _{p,t} = utility($makespan_{p,t}$) – price($Is_{p,t}$)
5. Compute surplus_{best} as maximum of surplus _{p,t}
6. If (surplus_{best} > surplus_{current allocation})
 - bid = ($\$_{p,t}, makespan_{p,t}, Is_{p,t}$) where surplus_{best} = surplus _{p,t}
 - send bid to market
7. Go to 2.

Processor Agent *pa* Process

1. Get current prices and own allocation
2. Compute surplus_{current allocation}
3. For all time slots $t \in 1.. T_{max}$:
 - Compute surplus_t = utility(*t*) – price(*t*)
4. Compute surplus_{best} as maximum of surplus_t
5. If (surplus_{best} > surplus_{current allocation})
 - bid = ($\$_{t}$, *t*) where surplus_{best} = surplus_t
 - Send bid to market
6. Go to 1.

First Results

If processors are easily available (cheep)

- Global scheduling outperforms market based scheduling
- wrt. surplus as optimization goal

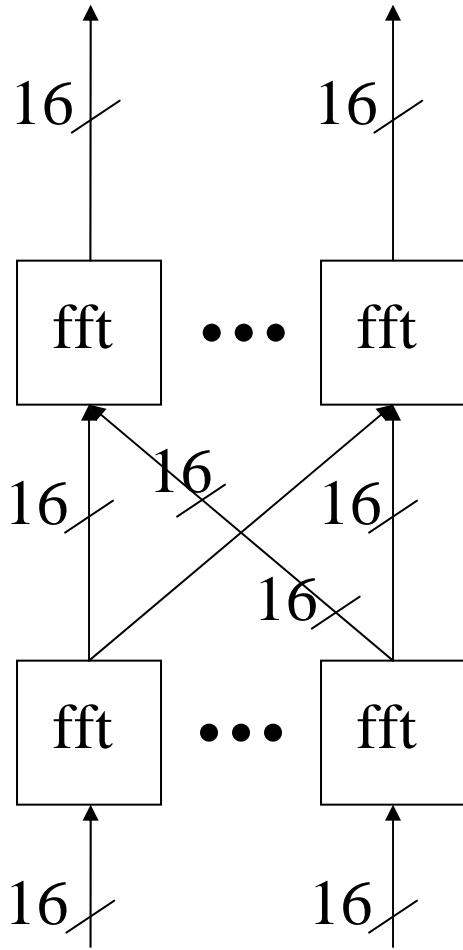
If processors limited resource

- Market based scheduling outperforms global scheduling wrt. surplus
- May not allocate all tasks

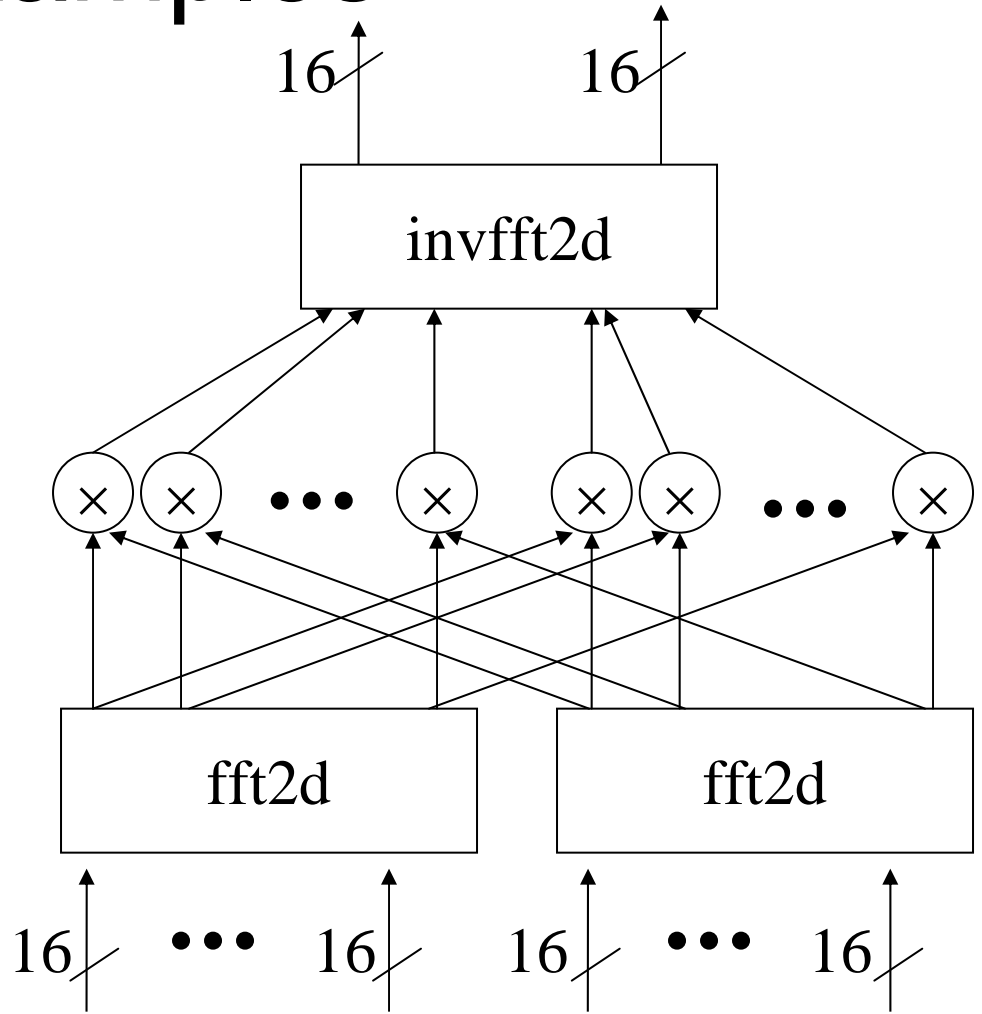
Concussion

- Scheduling helps
 - focusing on problems at hand instead of implementations especially optimizations
 - managing change
 - to sensor network
 - in programs and their configuration
 - in the set of programs deployed
- Challenges
 - Adaptation to sensor-network specifics
 - Test in real program simulation / in the field

Examples



fft2d / invfft2d



convolution

