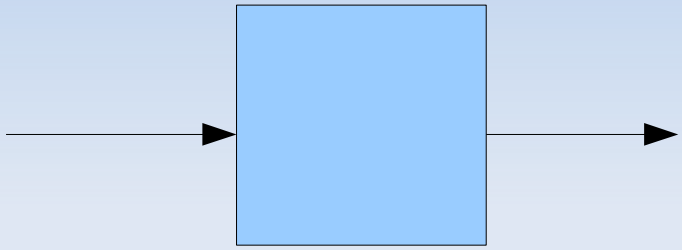
Modelica Multi-core Parallel Simulation using OpenMP and Optional Decoupling Elements

Martin Sjölund <martin.sjolund@liu.se>
Peter Fritzson <peter.fritzson@liu.se>
Linköping University, Sweden

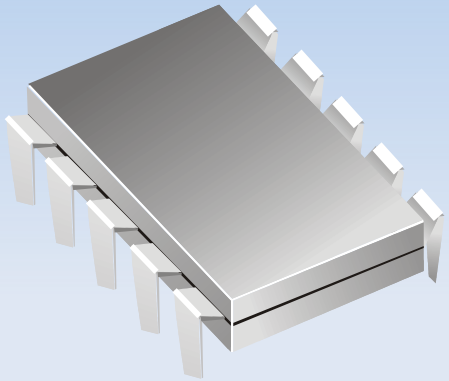
OpenModelica Workshop
Feb 2012, Linköping University, Sweden

Symptom: Simulation is slow
Why?

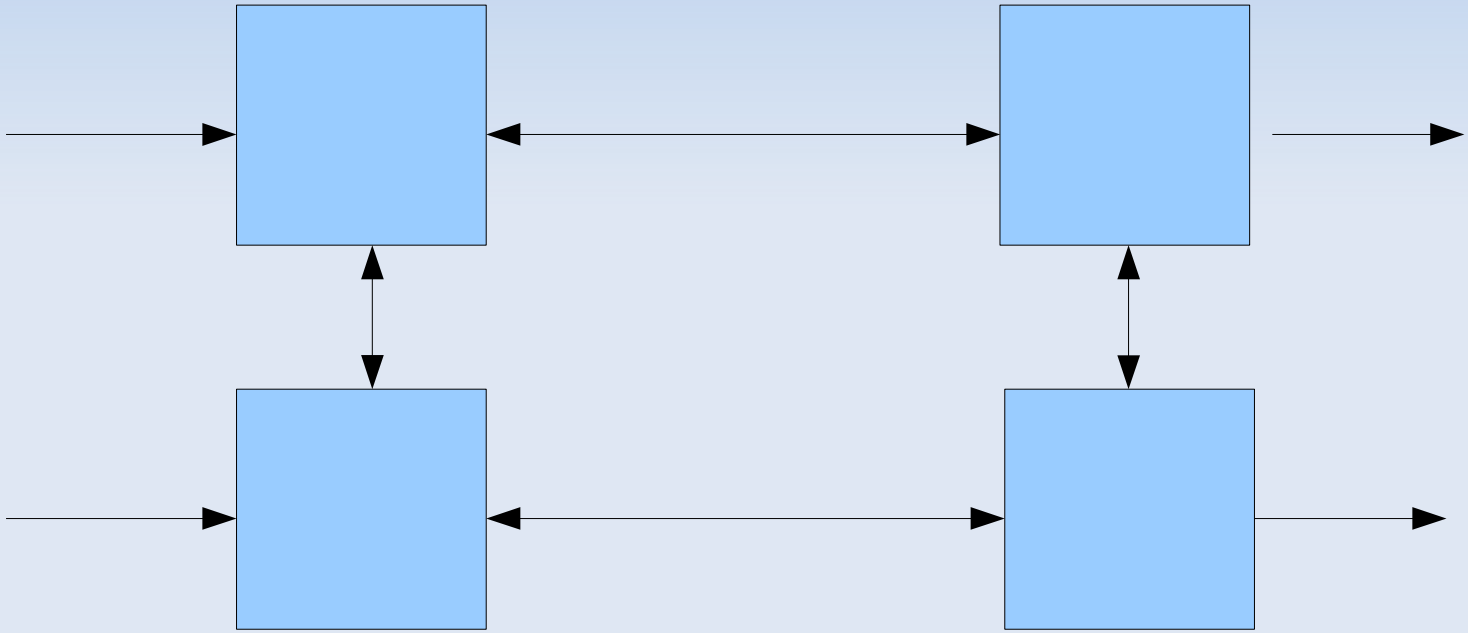
Simple Model (10 years ago)



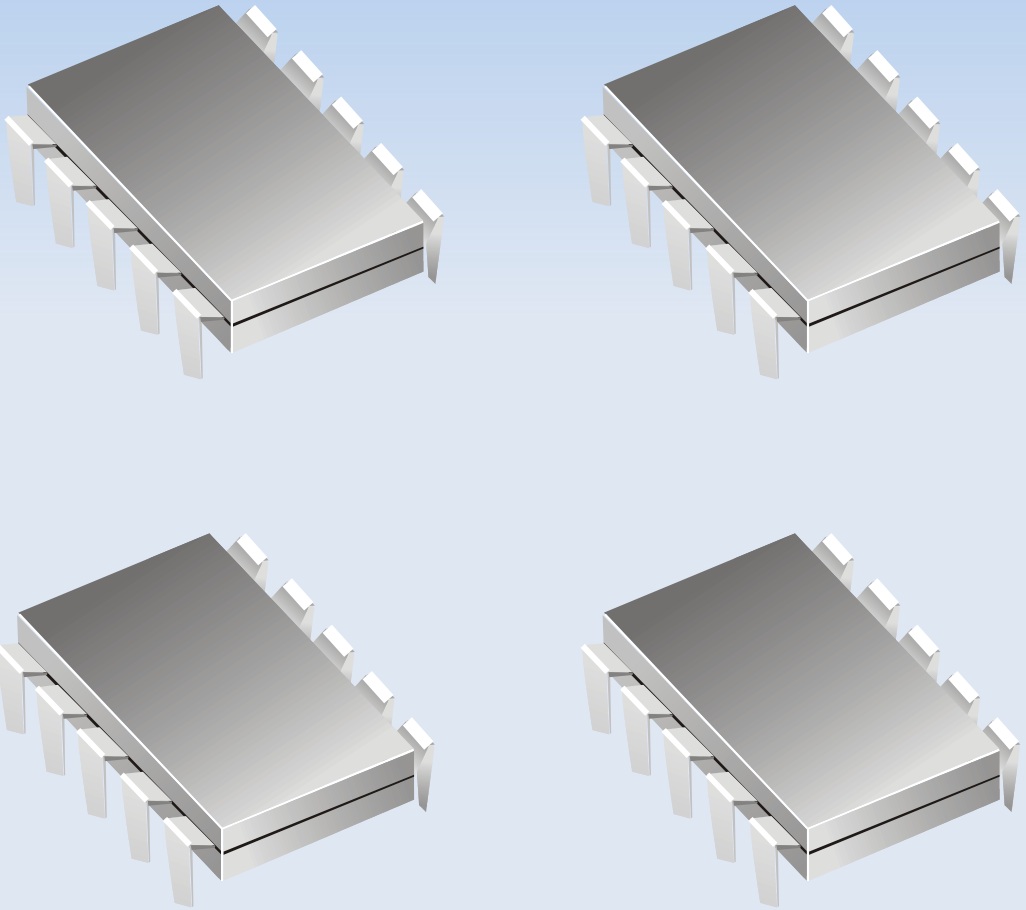
Simple Computer (10 years ago)



Complex Model (Today)



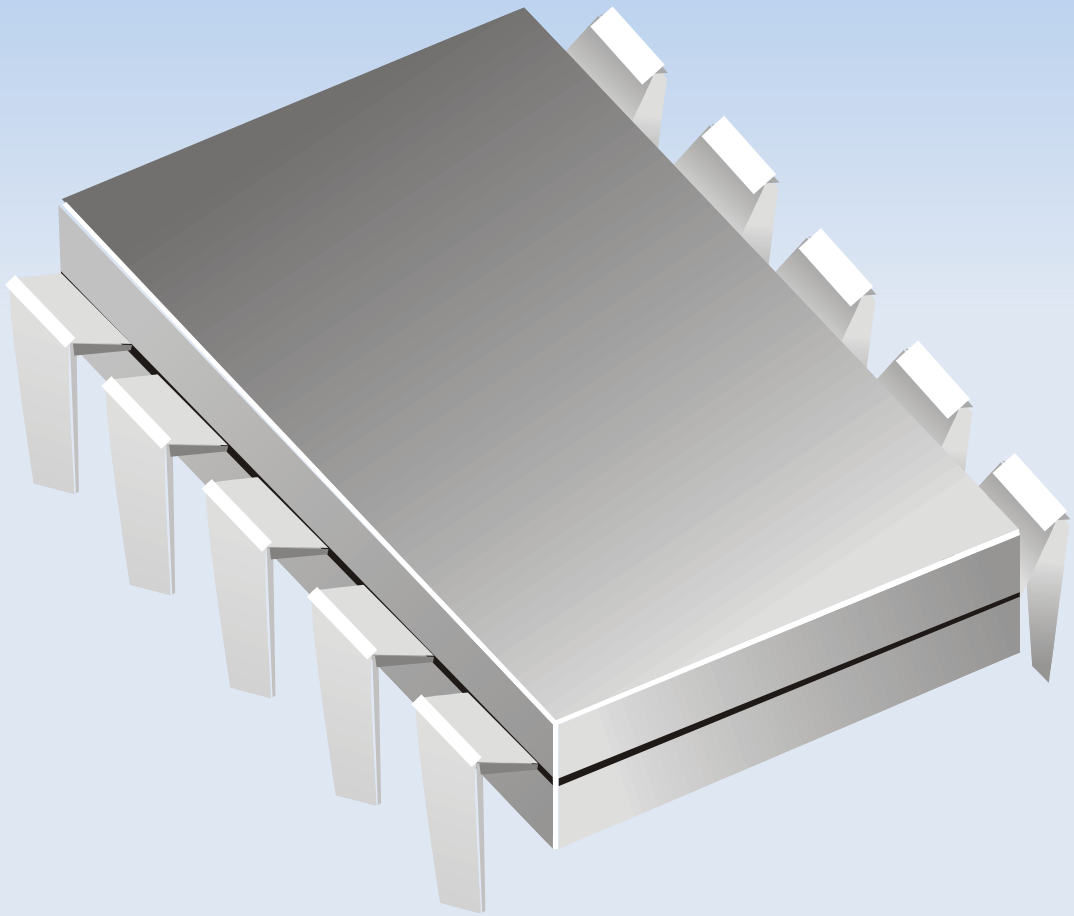
Computers Today



The Problem

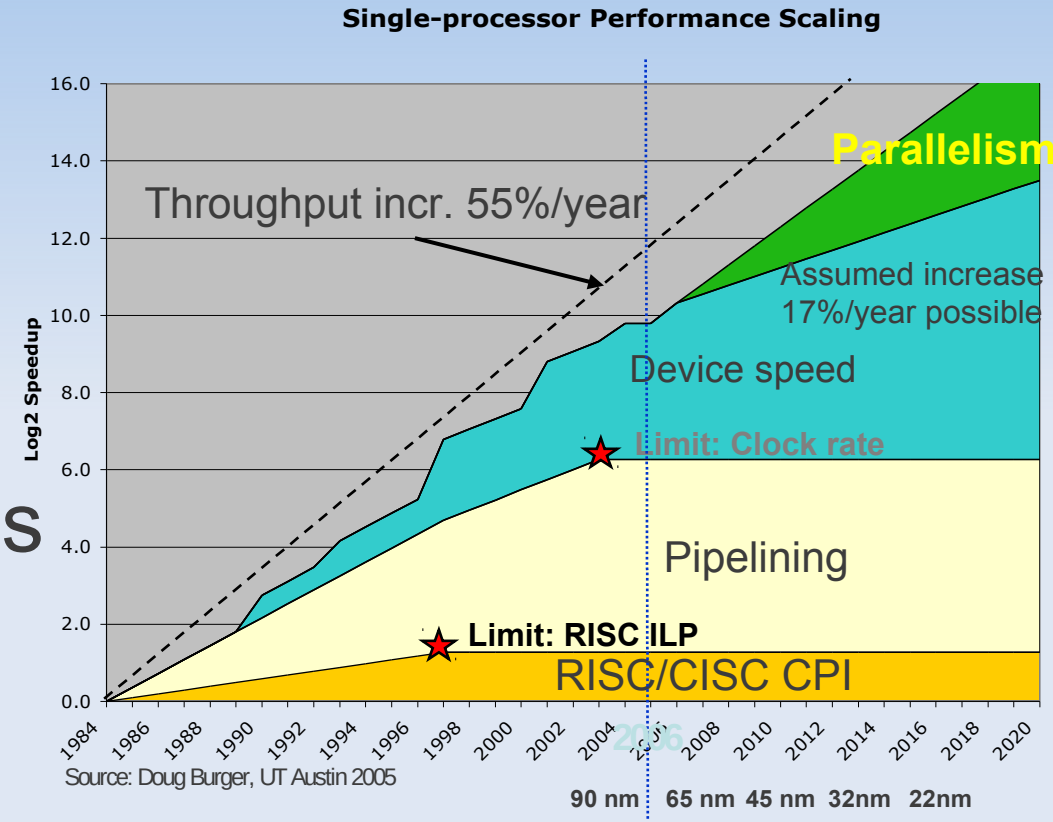
- Algorithms for numerical simulation
  - Mostly designed for single CPUs
  - Scaled well until we got multi-core CPUs
  - Not much research to parallelize simulations

Computer We Want Today

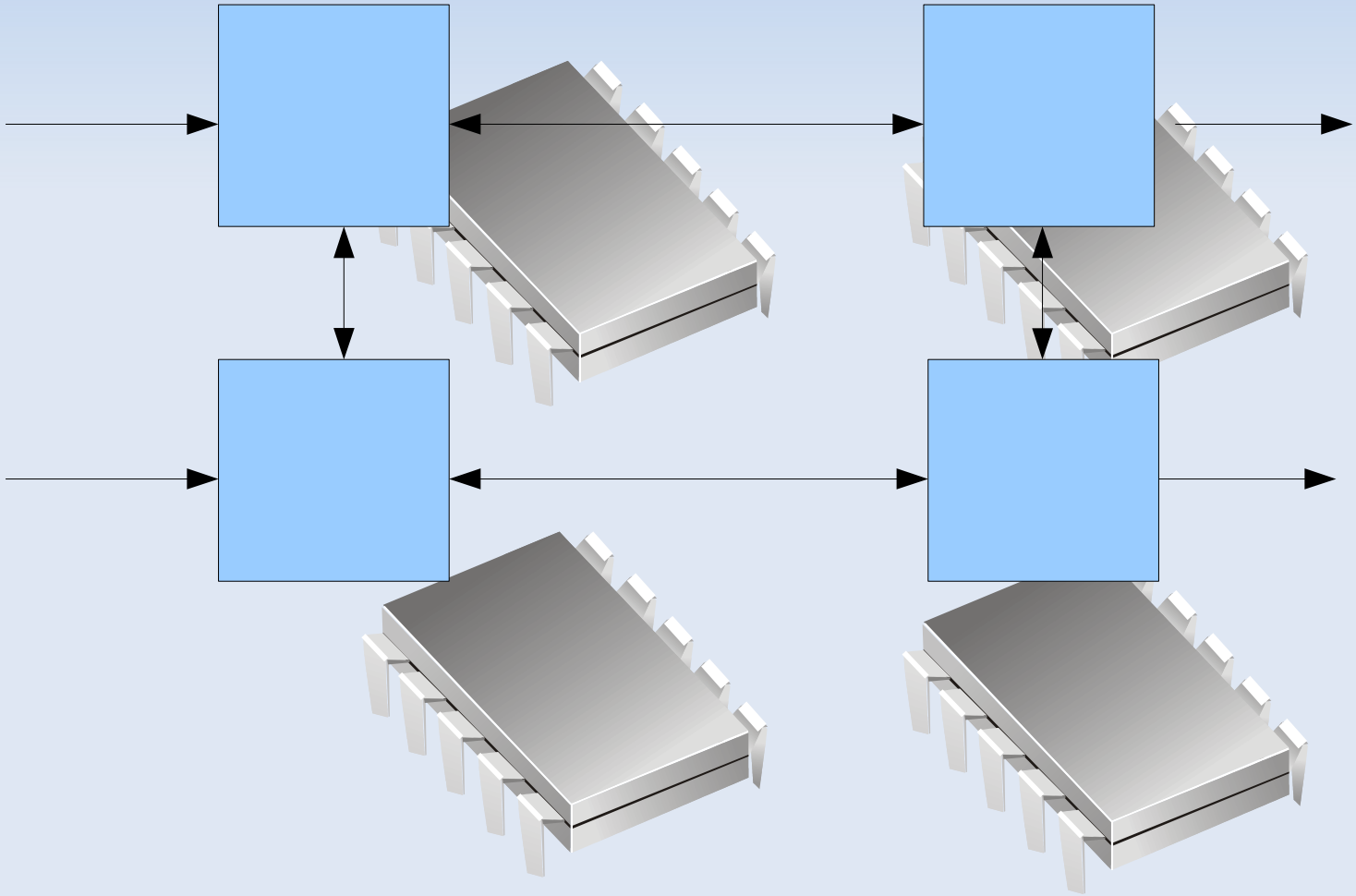


Modern computation units

- Multi-Core is the standard even for home users
- Penalizes single-threaded applications
- Stuck with CPU performance from 2004



Idea: Map Submodels to CPUs



Partitioning Algorithm

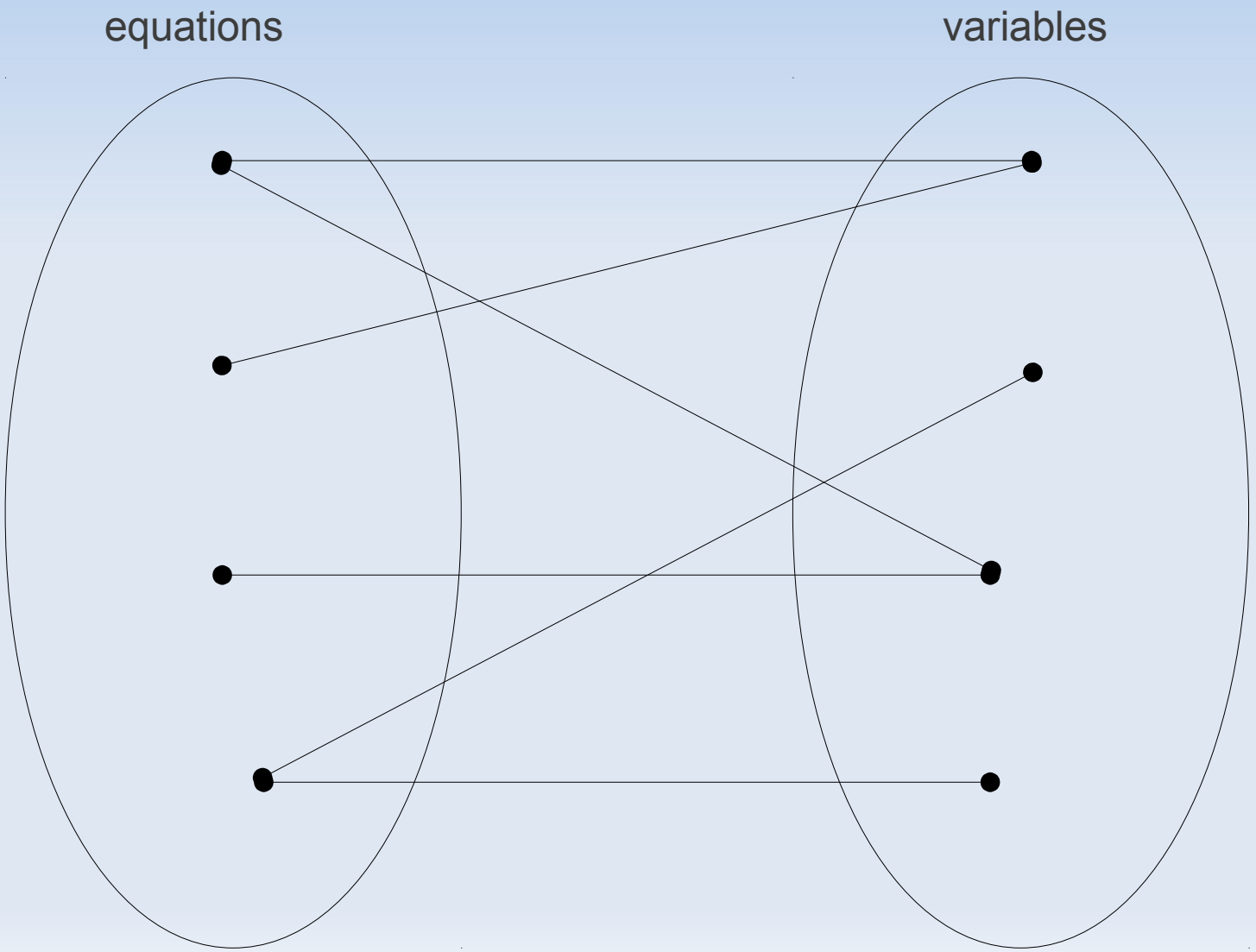
Incidence Matrix

	*					
*						
					*	
	*		*		*	
		*			*	
*				*		
*				*		*

Incidence Matrix

- Adjacency of variables/equations
- The matrix represents an undirected graph
- We want to find the trees in this graph
  - Apply standard graph algorithm

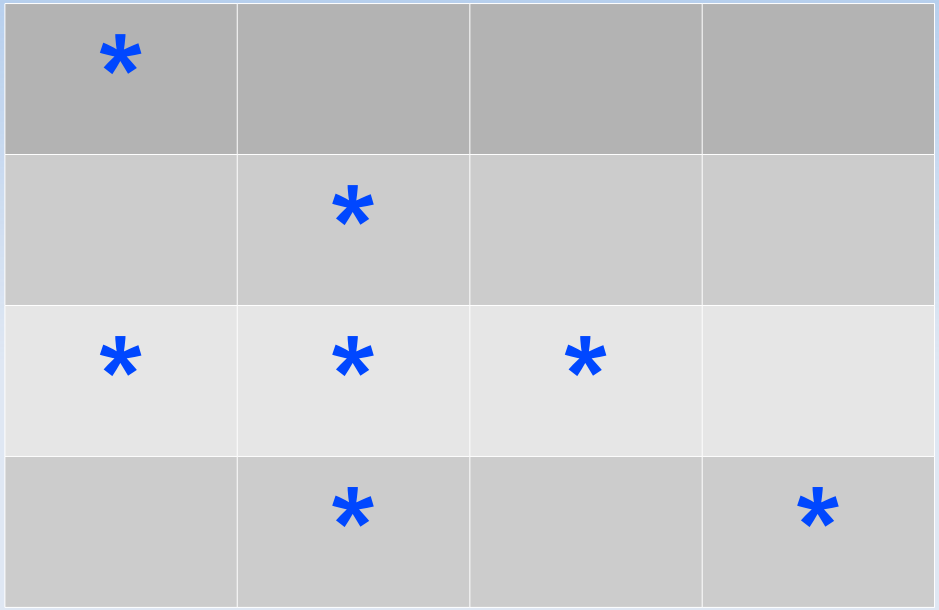
Bipartite Graph

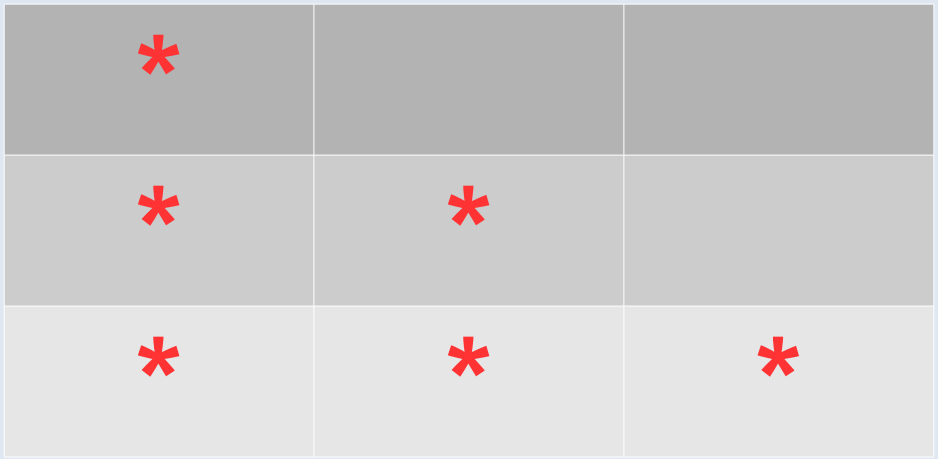


Found Independent Systems

	*					
*						
					*	
	*		*		*	
		*			*	
*				*		
*				*		*

Sorted Systems





Using the Partitioned System

Backend

- Many algorithms scale non-linearly
  - Having smaller systems speeds things up

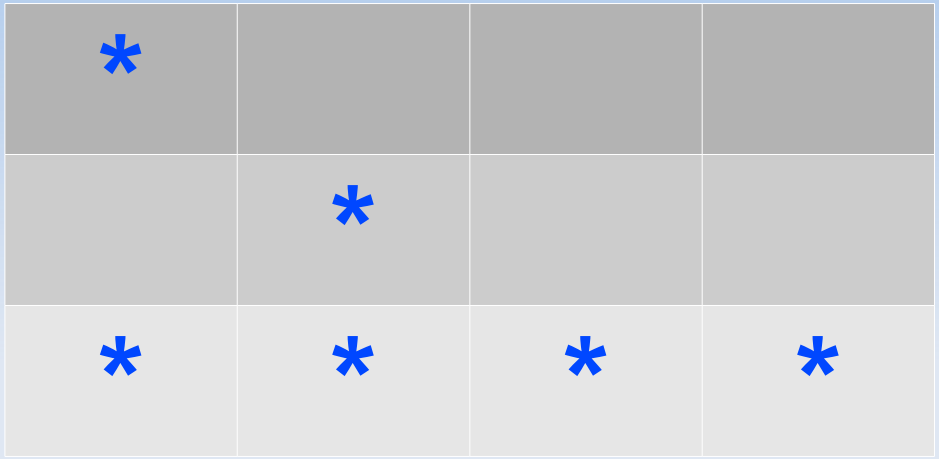
Code Generation

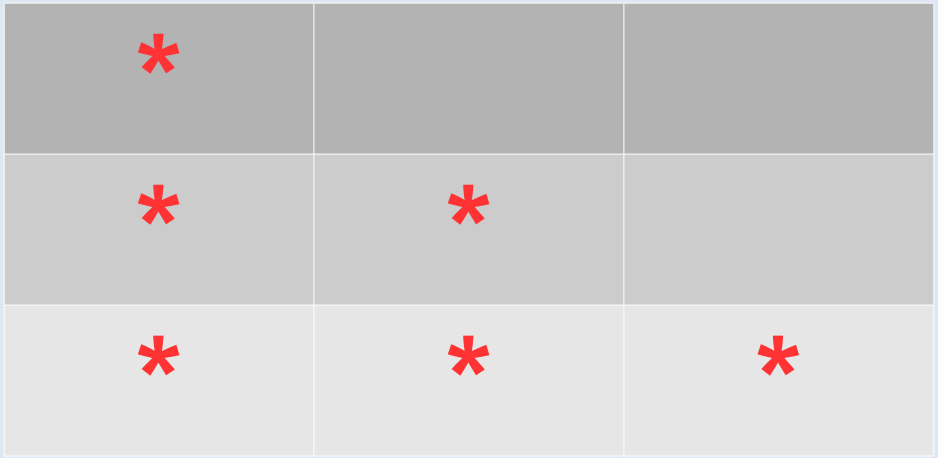
- Split ODE function into several smaller ones
  - GCC performs better due to complexity with large functions and optimization
  - Can be trivially parallelized
- Static scheduling
  - Requires calculating expected or worst-case runtime of each system

```
#pragma omp parallel for
private(id,th_id)
schedule(static)

for (id=0; id<2; id++) {

  th_id =
omp_get_thread_num();

  functionODE_systems[id]
(data,th_id);

}
```

Secondary Uses

Non-square systems





How Well Does It Work?

Speed-Up

- Perfectly balanced subsystems
  - Linear scaling with number of CPUs
  - Are very rare

Limitations

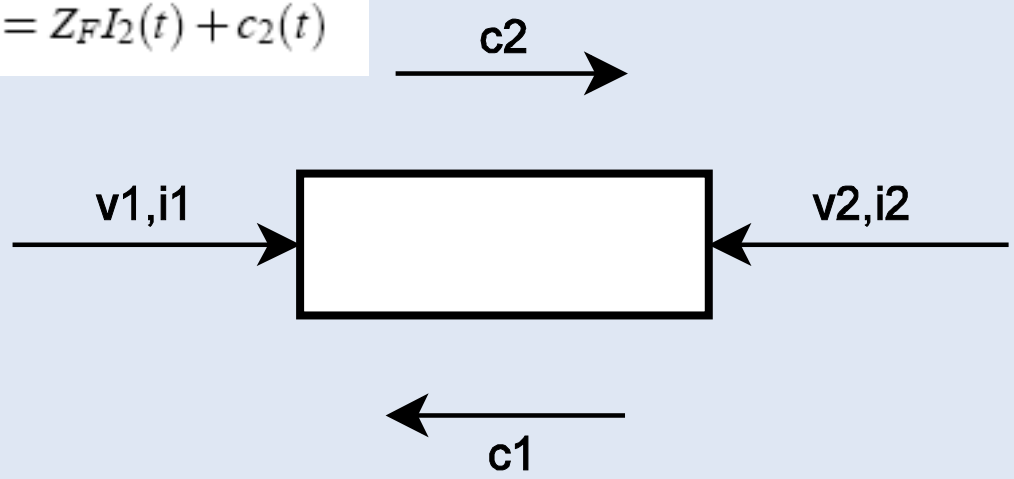
- Requires models with independent submodels
  - In Modelica, connections using `pre()` or `delay()`
- Most models are very strongly coupled

Transmission Line Modeling

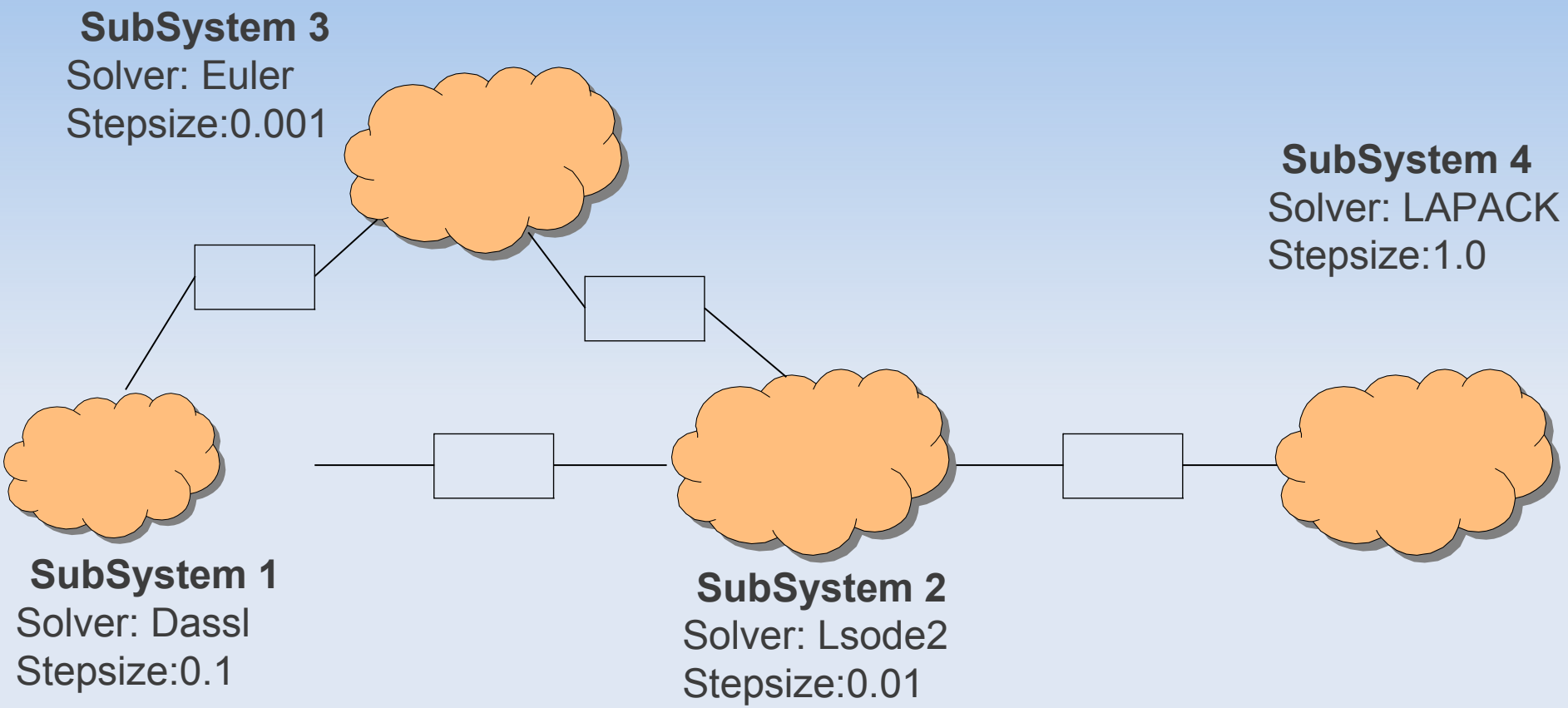
- TLM – Transmission Line Modeling – numerically stable co-simulation
- Physically motivated time delays are inserted between components
- Originally used in hydraulics with propagation delays along pipes
- Generalized to other engineering domains

$$c_1(t) = V_2(t - T_{tlm}) + Z_F I_2(t - T_{tlm})$$
$$c_2(t) = V_1(t - T_{tlm}) + Z_F I_1(t - T_{tlm})$$
$$P_1(t) = Z_F I_1(t) + c_1(t)$$
$$P_2(t) = Z_F I_2(t) + c_2(t)$$

c1, c2 are the TLM-parameters
Ttlm is the information propagation time
Zf is the implicit impedance



Distributed model



Future Work

- Parallelize algebraic part of the system
  - Only the ODE so far
- Test system on larger models
  - Using TLM connectors to take advantage of parallelism using the `delay()` operator in Modelica