A Strategy for Parallel Simulation of Declarative Object-Oriented Models of Generalized Physical Networks

Francesco Casella
(francesco.casella@polimi.it)

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Introduction and motivation

- Moore's law depends on multi-core architectures since 2007
- Declarative O-O modelling is now a mature & established field (1997: Modelica 1.0)
Introduction and motivation

- Moore's law depends on multi-core architectures since 2007
- Declarative O-O modelling is now a mature & established field (1997: Modelica 1.0)

however

- Simulation code generated by state-of-the-art Modelica tools as of 2012 is still fully sequential!

why?
TLM Modelling

- Main idea: physical interactions given by wave propagation phenomena with finite delay
  - pressure / flow waves in hydraulic circuits
  - electromagnetic waves in transmission lines
  - elastic waves in mechanical systems
- Explicitly partition the system with Transmission Line Models

\begin{align*}
e_1(t) &= Zf_1(t) + e_2(t - T_{tl}) + Zf_2(t - T_{tl}) \\
e_2(t) &= Zf_2(t) + e_1(t - T_{tl}) + Zf_1(t - T_{tl})
\end{align*}
TLM Modelling

- Main idea: physical interactions given by wave propagation phenomena with finite delay
  - pressure / flow waves in hydraulic circuits
  - electromagnetic waves in transmission lines
  - elastic waves in mechanical systems
- Explicitly partion the system with Transmission Line Models

\[
e_1(t) = Zf_1(t) + e_2(t-T_{tl}) + Zf_2(t-T_{tl})
\]
\[
e_2(t) = Zf_2(t) + e_1(t-T_{tl}) + Zf_1(t-T_{tl})
\]

Sub-systems can be solved independently (→ in parallel) for T seconds
TLM Modelling

Advantages

• Modelling is physically accurate – no approximations

Disadvantages

• Decoupling elements must be inserted manually
• Physical delays $T$ are usually very small
• Maximum step size constrained by $T$

requires expertise

parallel simulation might be slower than sequential simulation of the coupled model
Peter Aronsson's PhD work

\[ F(x, \dot{x}, v, t) = 0 \]

\[ \dot{x} = f(x, t) \]
\[ v = g(x, t) \]

explicit assignments
+ implicit equations to solve
Advantages

- Applies to any equation-based model without manual intervention
- Optimal scheduling possible in principle

Disadvantages

- Accurate estimation of computation and communication delays crucial
- Potentially bad performance if delays not correct
- Task merging and clustering algorithms very involved
- Possibly too complex for large systems

never found way into production tools
Goals of this work

Identify a (large) class of O-O models that have a special structure of the incidence matrix

Propose an algorithm which is simple to implement, but provides near optimal performance for that class of models
Main Idea
Main Idea

Exploit for parallelization!
Proposed algorithm

1. Build E-V digraph
2. Find a complete matching
   (possibly using Pantelides / Dummy Derivatives for higher index systems)
3. Replace non-matching edges with $E \rightarrow V$ arc; collapse V-nodes with matching E-nodes (a directed graph is obtained)
4. Run Tarjan's algorithm and identify strong components
   (systems of equations to be solved simultaneously)
Proposed algorithm

1. Build E-V digraph
2. Find a complete matching
   (possibly using Pantelides / Dummy Derivatives for higher index systems)
3. Replace non-matching edges with E→V arc; collapse V-nodes with matching E-nodes (a directed graph is obtained)
4. Run Tarjan's algorithm and identify strong components
   (systems of equations to be solved simultaneously)
5. Collapse each strong component into a single macro-node
6. Let $i = 1$
7. Search for all sinks and collect them in set $S_i$
   (they correspond to equations that can be solved independently)
8. Delete all nodes in set $S_i$ and all associated arcs from the graph
9. If there are nodes left, increase $i$ by 1 and goto 6.
Proposed algorithm

1. Build E-V digraph
2. Find a complete matching
   (possibly using Pantelides / Dummy Derivatives for higher index systems)
3. Replace non-matching edges with E→V arc; collapse V-nodes with matching E-nodes (a directed graph is obtained)
4. Run Tarjan's algorithm and identify strong components
   (systems of equations to be solved simultaneously)
5. Collapse each strong component into a single macro-node
6. Let $i = 1$
7. Search for all sinks and collect them in set $S_i$
   (they correspond to equations that can be solved independently)
8. Delete all nodes in set $S_i$ and all associated arcs from the graph
9. If there are nodes left, increase $i$ by 1 and goto 6.

At termination, all equations are collected in the sets $S_i$
Equations in $S_1$ can be solved independently, then equations in $S_2$, etc.
Generalized networks

- Storage components: storage of physical quantities, represented by state variables (known at each time step!)
- Flow components: describe the flow of the quantity, based on the boundary values

- V-F-V topology: no implicit equations

- V-F-F-V topology: implicit equations (macro-nodes)
Example 1: Thermal Networks

Storage components

\[ C(T_i) \frac{dT_i}{dt} = \sum_j Q_{i,j} \]

Flow components

\[ Q_i = G_i (T_i,a - T_i,b) \]
Example 1: Application of the algorithm

\[ Q_i = G_i(T_{i,a} - T_{i,b}) \]
Example 1: Application of the algorithm

\[ C(T_i) \frac{dT_i}{dt} = \sum_j Q_{i,j} \]

\[ Q_i = G_i (T_{i,a} - T_{i,b}) \]
Example 1: Application of the algorithm

\[ C(T_i) \frac{dT_i}{dt} = \sum_j Q_{i,j} \]

\[ Q_i = G_i(T_{i,a} - T_{i,b}) \]

Always 2 sets $S_i$ regardless of network dimension!
Example 2: Thermo-Hydraulic Networks

Storage components

\[
\begin{bmatrix}
e_i & h_i & \rho_i & \frac{\partial \rho_i}{\partial p} & \frac{\partial \rho_i}{\partial T} & \frac{\partial e_i}{\partial p} & \frac{\partial e_i}{\partial T}
\end{bmatrix} = f(p_i, T_i)
\]

\[M_i = \rho_i V_i\]

\[\frac{dM_i}{dt} = \sum_j w_{i,j}\]

\[\frac{dE_i}{dt} = \sum_j w_{i,j} h_{i,j} + \sum_j Q_{i,j}\]

\[\frac{dM_i}{dt} = \frac{\partial \rho_i}{\partial p} \frac{dp_i}{dt} + \frac{\partial \rho_i}{\partial T} \frac{dT_i}{dt}\]

\[\frac{dE_i}{dt} = \left(\frac{\partial e_i}{\partial p} \frac{dp_i}{dt} + \frac{\partial e_i}{\partial T} \frac{dT_i}{dt}\right) M_i + e_i \frac{dM_i}{dt}\]

Flow components

\[Q_i = G_i (T_{i,a} - T_{i,b})\]

\[w_i = w(p_{i,a}, p_{i,b}, \rho_i)\]
Example 2: Thermo-Hydraulic Networks

Often takes up a large fraction of CPU time

Independent computation for each storage component

\[
\begin{bmatrix}
    e_i & h_i & \rho_i & \frac{\partial \rho_i}{\partial p} & \frac{\partial \rho_i}{\partial T} & \frac{\partial e_i}{\partial p} & \frac{\partial e_i}{\partial T}
\end{bmatrix} = f(p_i, T_i)
\]
Example 2: Thermo-Hydraulic Networks

\[ w_i = w(p_{i,a}, p_{i,b}, \rho_i) \]

\[ M_i = \rho_i V_i \]
Example 2: Thermo-Hydraulic Networks

\[
\frac{dE_i}{dt} = \sum_j w_{i,j} h_{i,j} + \sum_j Q_{i,j}
\]

\[
\frac{dM_i}{dt} = \sum_j w_{i,j}
\]
Example 2: Thermo-Hydraulic Networks

\[
\begin{align*}
\frac{dM_i}{dt} &= \frac{\partial \rho_i}{\partial p} \frac{dp_i}{dt} + \frac{\partial \rho_i}{\partial T} \frac{dT_i}{dt} \\
\frac{dE_i}{dt} &= \left( \frac{\partial e_i}{\partial p} \frac{dp_i}{dt} + \frac{\partial e_i}{\partial T} \frac{dT_i}{dt} \right) M_i + e_i \frac{dM_i}{dt}
\end{align*}
\]
Example 2: Thermo-Hydraulic Networks

Always 4 sets $S_i$ regardless of network dimension!
Task Scheduling

- Shared memory $\rightarrow$ negligible communication costs
  (beware of L1 caching issues!)

- The larger $\frac{\#(S_i)}{N_{\text{cores}}}$ is, the less the actual scheduling policy matters
Task Scheduling

- Shared memory $\rightarrow$ negligible communication costs (beware of L1 caching issues!)

- The larger $\frac{\#S_i}{N_{\text{cores}}}$ is, the less the actual scheduling policy matters
  - if many more independent tasks than cores, a first-come, first-served scheduling policy will result in all the cores running almost all the time

- Avoid waiting for the slowes one: start slowest tasks earlier
  - minimize chance of N-1 cores waiting for the Nth to complete its task

- Avoid overhead: merge several short tasks in a bigger one

- Only rough estimates of the running time are required
Task Scheduling

- Shared memory $\rightarrow$ negligible communication costs
  (beware of L1 caching issues!)

- The larger $\frac{\#(S)}{N_{\text{cores}}}$ is, the less the actual scheduling policy matters
  - if many more independent tasks than cores, a first-come, first-served scheduling policy will result in all the cores running almost all the time

- Avoid waiting for the slowest one: start slowest tasks earlier
  - minimize chance of N-1 cores waiting for the Nth to complete its task

- Avoid overhead: merge several short tasks in a bigger one

- Only rough estimates of the running time are required

- Set up threads at the beginning of the simulation, activate them @ each time step
Conclusions

- Exploiting Moore's law from 2008 requires exploiting parallelism
- Strategies for parallel simulation of O-O declarative models exist, not implemented in mainstream tools yet (implementation difficulties, not enough cores to justify the overhead)
- A very simple algorithm has been proposed in this work
- It can provide nearly optimal parallel allocation of resources in generalized network models
- Only very rough estimates of task execution times are needed
- Other parallelization strategies could be implemented together with this one to improve overall performance
  - parallel numerical computation of Jacobians
  - parallel solution of large implicit systems in stiff solvers
  - ...
- Beware of thread switching overhead and L1 caching issues!
Thank you for your kind attention!