

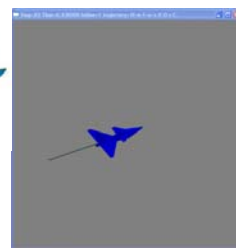
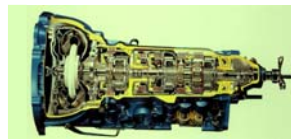
Automatic Parallelization of Simulation Code for Equation-Based Models with Software Pipelining and Measurements on Three Platforms

Håkan Lundvall, Kristian Stavåker,
Peter Fritzson, Christoph Kessler

*PELAB – Programming Environments Laboratory Dept. of Computer and
Information Science Linköping University, SE-581 83 Linköping, Sweden
{haklu, krsta, petfr, chrke}@ida.liu.se*

Examples of Complex Systems

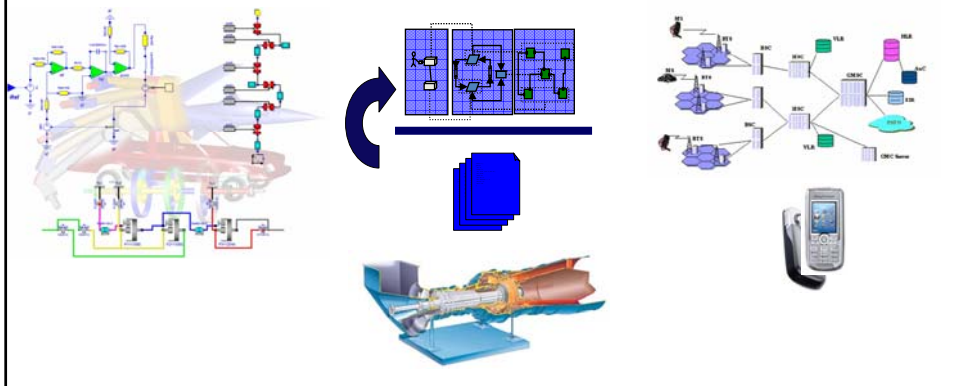
- Robotics
- Automotive
- Aircraft
- Living Organisms
- Power Plants
- Heavy Vehicles



Advertisement: MODPROD'2009 Workshop and Tutorials on Model-Based Product Development

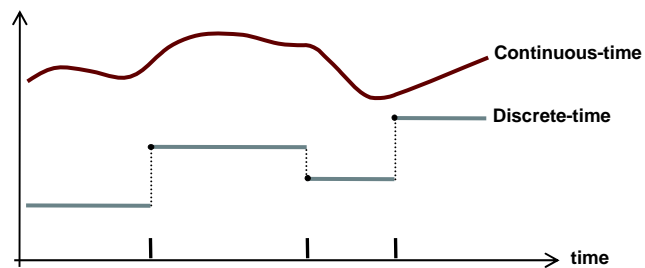
www.modprod.liu.se

Linköping, February 3, 4 2009
Keynote: Richard Soley, CEO OMG



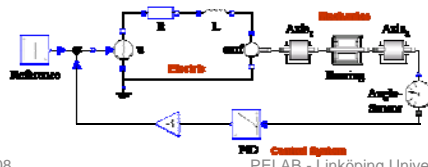
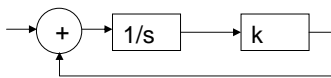
Types of Systems

- Dynamic/static systems
- Continuous/discrete-time systems
- Hybrid systems



Continuous-Time Modeling and Simulation Approaches

- Hard-coding using some programming language (Fortran, C++, ...)
- Use a block-based simulation program (SIMULINK)
- Use an equation-based language (Modelica, gPROMS, VHDL-AMS)



```
cout.setf(ios::scientific);
cout.precision(12);
for (i = 0; i < steps; ++i) {
    for (pp = 0; pp < N; ++pp) {
        for (j = 0; j < 4; ++j) {
            struct timeval & tv = timings[i][pp][j];
            double t = (tv.tv_sec-t_start.tv_sec)+0.000001*(tv.tv_usec-t_start.tv_usec);
            cout << t << char(9);
        }
        cout << endl;
    }
    struct timeval & tv = timings[steps-1][0][0];
    double t = (tv.tv_sec-t_start.tv_sec)+0.000001*(tv.tv_usec-t_start.tv_usec);
```

12/18/2008

PELAB - Linköping University

5

Equation-based Object-oriented Modeling Languages

Modelica

12/18/2008

PELAB - Linköping University

6

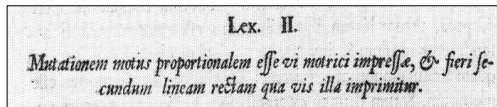
The Form – Equations

- Equations were used in the third millennium B.C.
- Equality sign was introduced by Robert Recorde in 1557

14.ze. — + — .15.9 — = — = 71.9.

Newton still wrote text (Principia, vol. 1, 1686)

“The change of motion is proportional to the motive force impressed”



CSSL (1967) introduced a special form of “equation”:

variable = expression

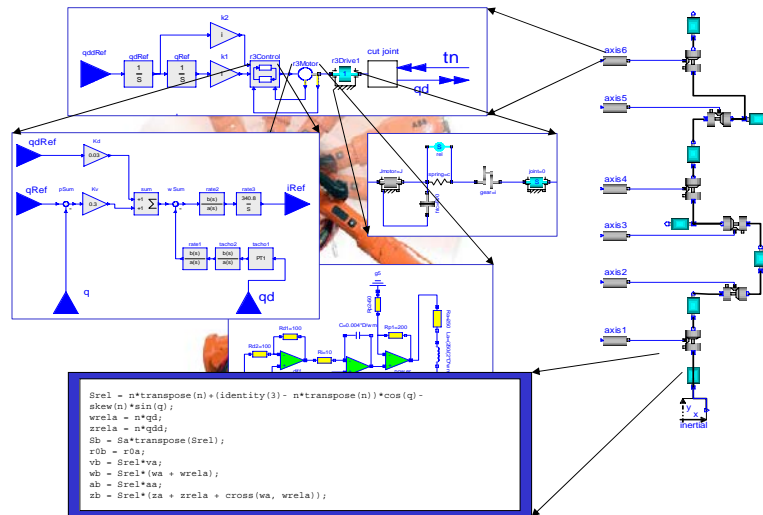
$v = \text{INTEG}(F) / m$

Programming languages usually do not allow equations!

Modelica – The Next Generation Modeling Language

- Declarative language
 - Equations and mathematical functions allow acausal modeling, high level specification, increased correctness
- Multi-domain modeling
 - Combine electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc...
- Everything is a class
 - Strongly typed object-oriented language with a general class concept, Java & MATLAB-like syntax
- Visual component programming
 - Hierarchical system architecture capabilities
- Efficient, non-proprietary
 - Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations, ~150 000 lines on standard PC

Hierarchical System Decomposition into Components – Industry Robot



Courtesy of Martin Otter and Hilding Elmqvist
12/18/2008

PELAB - Linköping University

9

Key Concept: Acausal Modeling with Equations

- What is *acausal* modeling/design?
- Why does it increase *reuse*?
 - The *acausality* makes Modelica library classes *more reusable* than traditional classes containing assignment statements where the input-output causality is fixed.
- Example: a resistor equation: $R \cdot i = v$;
 ... can be used in three ways:
 - $i := v/R$;
 - $v := R \cdot i$;
 - $R := v/i$;

12/18/2008

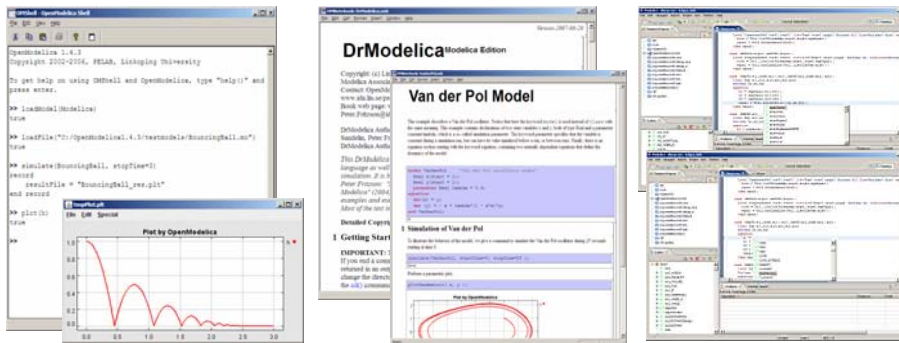
PELAB - Linköping University

10

What is OpenModelica?

www.openmodelica.org

- **Advanced Interactive Modelica compiler (OMC)**
 - Supports most of the Modelica Language
- **Basic environment for creating models**
 - OMShell – an interactive command handler * ModelicaML UML Profile
 - OMNotebook – a literate programming notebook * MetaModelica transforms
 - MDT – an advanced textual environment in Eclipse



12/18/2008

PELAB - Linköping University

11

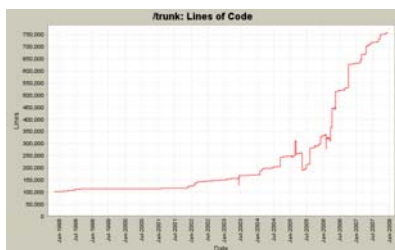
Open Source Modelica Consortium and Community

- **Open-source community services**
 - Website and Support Forum
 - Version-controlled source base
 - Bug database
 - Development courses

Open Source Modelica Consortium founded Dec 4, 2007

Now 11 Industry/Institute members and 7 university members.
17 individual members

Code Statistics

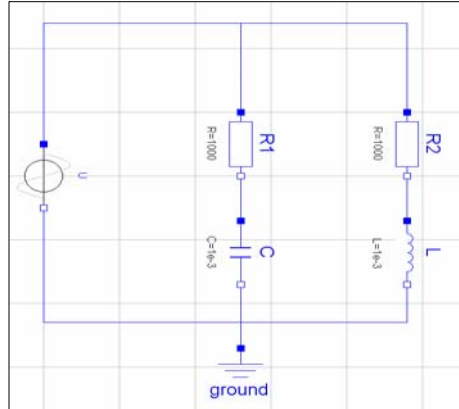


- Mature code base
- ~ 800k lines of code, doubled since 2005

- Bosch-Rexroth AG, Germany
- ABB Corporate Research AB, Sweden
- Siemens Industrial Turbomachinery, Sweden
- Equa Simulation AB, Sweden
- TLK Thermo, Germany
- VTT, Finland
- MostforWater, Belgium
- MapleSoft, Canada
- Emmeskay Inc., USA
- IFP, Paris, France
- MathCore Engineering AB

- Linköping University, Sweden
- Technical Univ of Hamburg-Harburg, Germany
- Technical Univ of Braunschweig, Germany
- Université Laval, Canada
- University of Queensland, Australia
- Griffith University, Australia
- Politecnico di Milano, Italy

A Simple Modelica Equation-Based Model



```

model Simple
  Real U,vc,vr1,vr2,vl,
        v2i,i1,i2;
  parameter Real
    C=1,R1=1,R2=1,L=1;
  equation
    U = 5*sin(time);
    U = vc + vr1;
    U = vl + vr2;
    i1 + i2 = i;
    der(vc)*C = i1;
    der(i2)*L = vl;
    vr1 = R1*i1;
    vr2 = R2*i2;
end Simple;
  
```

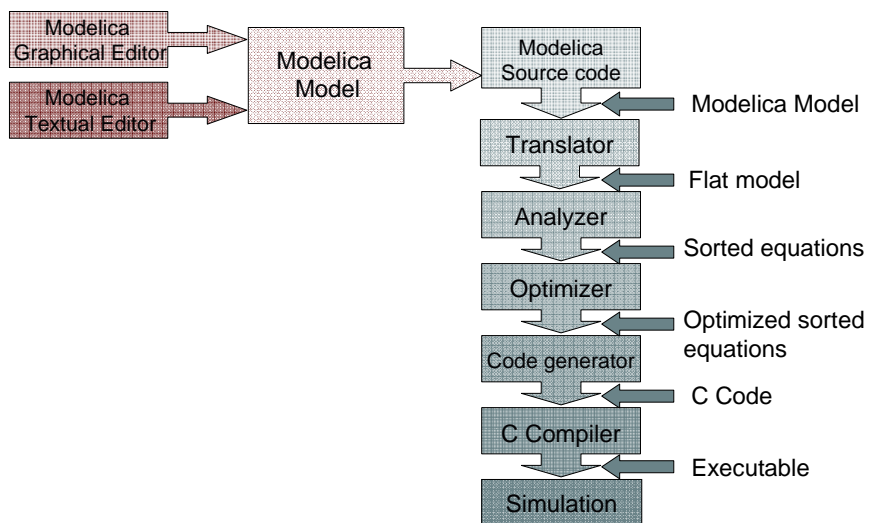
$x(t) = \{vc, i2\}$ State variables
 $y(t) = \{vr1, vl, vr2, i1, i\}$ Algebraic variables

Sorted equations in explicit form

```

U      := 5*sin(t)
vr2    := R2 * i2
vl     := U - vr2
der(i2) := vl / L
vr1    := U - vc
i1     := vr1 / R1
der(vc) := i1 / C
i      := i1 + i2
  
```

Model Translation Process to Differential Algebraic Equations to Code



Integrating Parallelism and Mathematical Models

Three Parallelization Approaches

- **Automatic Parallelization of Mathematical Models**
 - Parallelism over the numeric solver method
 - Parallelism over time
 - *Parallelism over the model equation system (this paper)*
- **Coarse-Grained Explicit Parallelization Using Components**
 - The programmer partitions the application into computational components using strongly-typed communication interfaces
 - Co-Simulation, Transmission-Line Modeling (TLM)
- **Explicit Parallel Programming**
 - Explicit parallel programming constructs within the *algorithmic* part of the modeling language

Parallelism over the Model Equation System

- Simulation = solution of DAEs/ODEs from models

$$\text{DAE} \begin{cases} g(\dot{X}, X, Y, t) = 0 \\ h(X, Y, t) = 0 \end{cases} \quad \text{ODE} \quad \dot{X} = f(X, Y, t)$$

- In each step of numerical solver:
 - Calculate \dot{X} and Y
- Parallelization approach: perform the **calculation** of \dot{X} in **parallel** (calculate “the right-hand side”, f in parallel)
 - Called *parallelization over the equation system*
- Drawback: Numeric solver might become bottle-neck
- Inlining and **parallelization of solver** can remove this bottle-neck

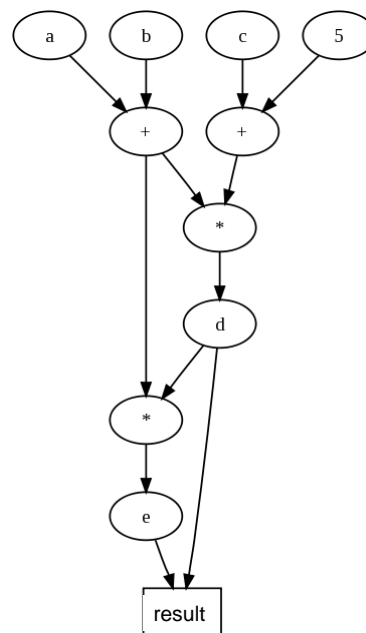
12/18/2008

PELAB - Linköping University

17

Data Flow Task Graph of Right-hand Side

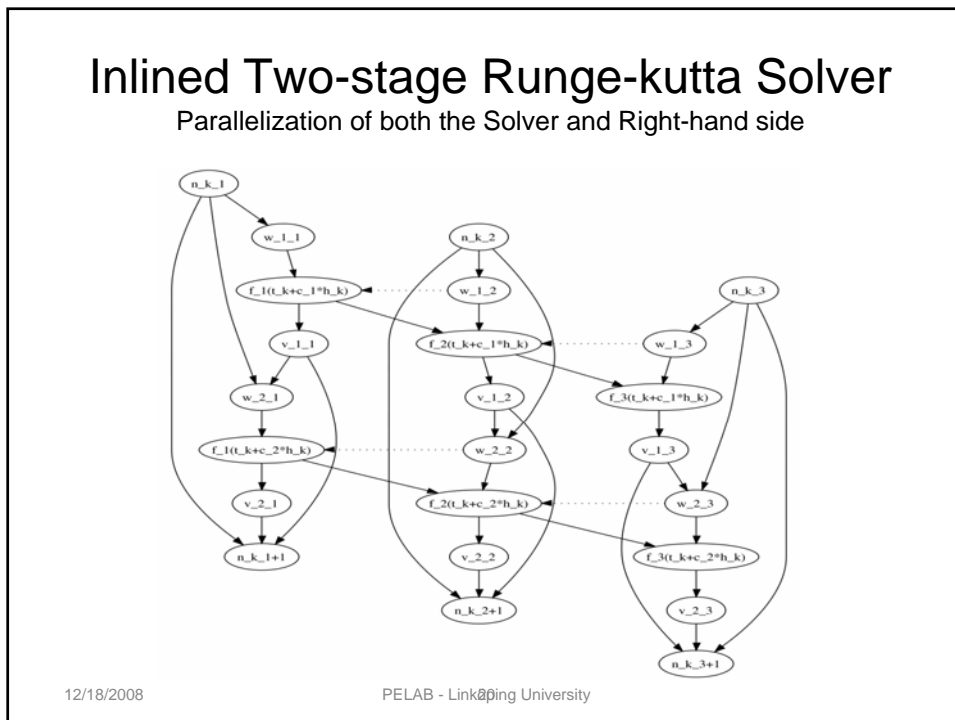
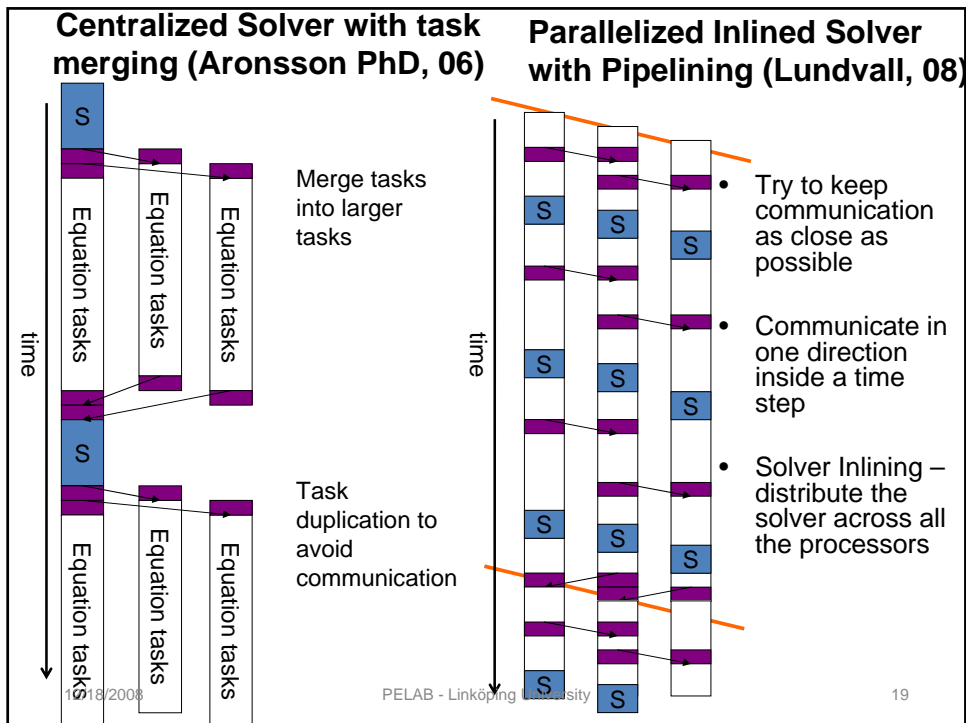
- Nodes represent values
 - Variables
 - Constants
 - Operands
- Arcs represent dependencies
- $d := (a + b) * (c + 5)$
- $e := (a + b) * d$



12/18/2008

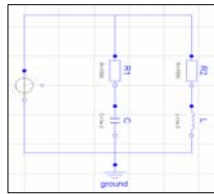
PELAB - Linköping University

18



Translation of Mathematical Models

Compilation of OO Equation-Based Models to Code



Hierarchical model

Flat model

Optimized and sorted equations

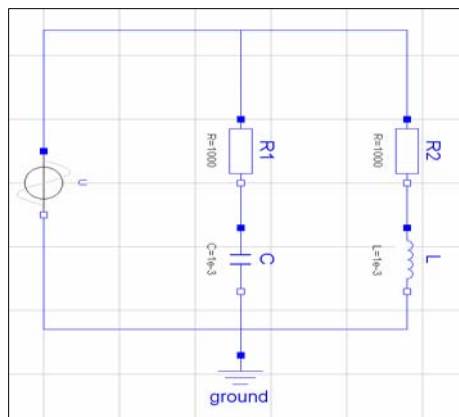
Program code

```
U = 5*sin(time);
U = vc + vr1;
U = v1 + vr2;
i1 + i2 = i;
der(vc)*C = i1;
der(i2)*L = v1;
vr1 = R1*i1;
vr2 = R2*i2;
```

```
U := 5*sin(t)
vr2 := R2 * i2
v1 := U - vr2
der(i2) := v1 / L
vr1 := U - vc
i1 := vr1 / R1
der(vc) := i1 / C
i := i1 + i2
```

```
for (i = 0; i < steps; ++i) {
  for (pp = 0; pp < N; ++pp) {
    for (j = 0; j < 4; ++j) {
      struct timeval & tv = ti
      double t = (tv.tv_sec-t_s
      cout << t << char(9);
    }
  }
  cout << endl;
}
struct timeval & tv = timings[ste
table t = (tv.tv_sec-t_start.tv_s
```

A Simple Modelica Model



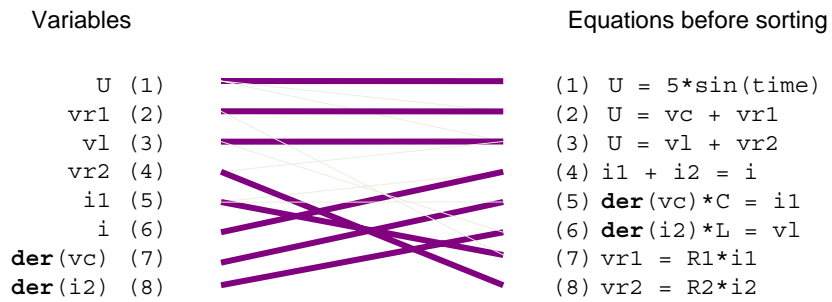
$x(t) = \{vc, i2\}$ State variables
 $y(t) = \{vr1, v1, vr2, i1, i, U\}$ Algebraic variables

$0 =$

```
model Simple
  Real U,vc,vr1,vr2,v1,
    v2i,i1,i2;
  parameter Real
    C=1,R1=1,R2=1,L=1;
equation
  U = 5*sin(time);
  U = vc + vr1;
  U = v1 + vr2;
  i1 + i2 = i;
  der(vc)*C = i1;
  der(i2)*L = v1;
  vr1 = R1*i1;
  vr2 = R2*i2;
end Simple;
```

```
U - 5*sin(time)
U - (vc + vr1)
U - (v1 + vr2)
i1 + i2 - i
der(vc)*C - i1
der(i2)*L - v1
vr1 - R1*i1
vr2 - R2*i2
```

Sorting Equations in Data Dependency Order



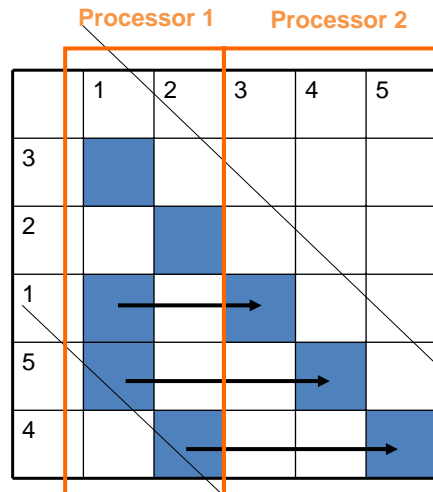
Block Lower Triangular Form (BLT) Representing sorted equations vs variables

		Variable number							
		1	4	3	8	2	5	7	6
Equation number	1	X							
	8		X						
	3	X	X	X					
	6			X	X				
	2	X				X			
	7					X	X		
	5						X	X	
	4						X		X

Can Access Distance be Reduced by Better Sorting?

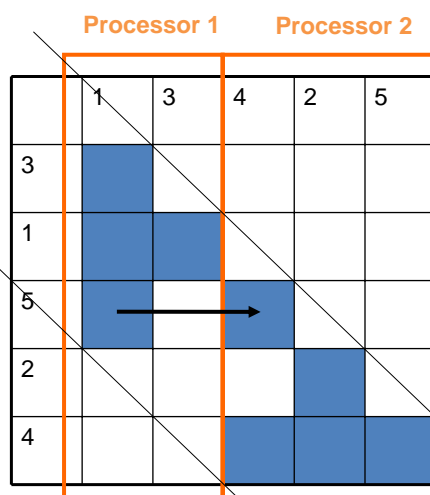
- There might exist **better sortings**, where the distance between the computation and the usage is reduced
- Here: 3 communications from left to right processor

Try to reduce that by better sorting

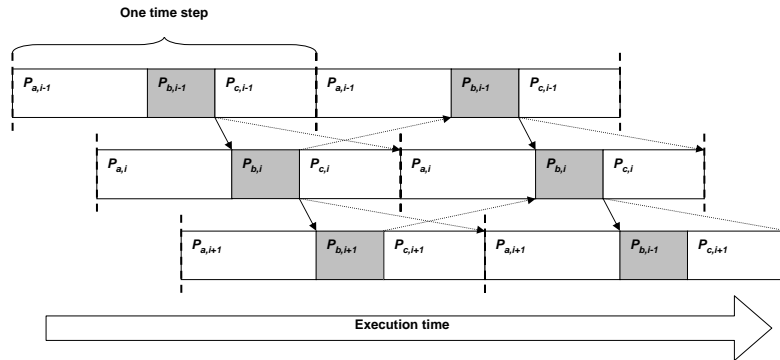


Extra Sorting Step Performed

- **Move equations upward** without violating the BLT-form
- Gives less communication
- In example: Only 1 communication arrow now left



Pipelining Groups of Processes

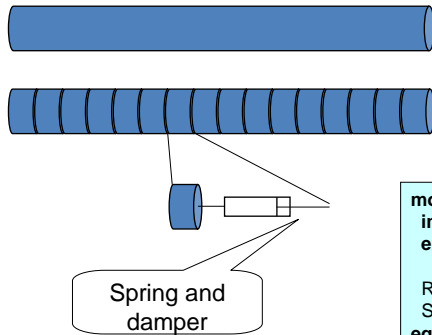


- In order to get a good, well-filled pipeline:
 - The execution time for the second group + communication time should be shorter than the calculation time for the other groups
 - The total execution time should be evenly distributed over the processes

Measurements on Different Architectures

Test Application Model – Nonlinear Mechanics

- Flexible Shaft, 100 elements



12/18/2008

PELAB -

```

model ShaftTest
  FlexibleShaft shaft(n=100);
  Modelica.Mechanics.Rotational.Torque src;
  Modelica.Blocks.Sources.Step c;
equation
  connect(shaft.flange_a,src.flange_b);
  connect(c.y,src.tau);
end ShaftTest;
  
```

```

model FlexibleShaft
  import Modelica.Mechanics.Rotational;
  extends Rotational.Interfaces.TwoFlanges;
  parameter Integer n(min=1) = 3;
  ShaftElement shaft[n];
equation
  for i in 2:n loop
    connect(shaft[i-1].flange_b, shaft[i].flange_a);
  end for;
  connect(shaft[1].flange_a,flange_a);
  connect(shaft[n].flange_b,flange_b);
end FlexibleShaft;
  
```

```

model ShaftElement
  import Modelica.Mechanics.Rotational;
  extends Rotational.Interfaces.TwoFlanges;
  Rotational.Inertia inertia1;
  SpringDamperNL springDamper1(c=5,d=0.11);
equation
  connect(inertia1.flange_b, springDamper1.flange_a);
  connect(inertia1.flange_a, flange_a);
  connect(springDamper1.flange_b, flange_b);
end ShaftElement;
  
```

Architectures

- Using Pthreads on Shared memory
 - Intel Xeon - 4 cores with hyperthreading (8 virtual cores)
 - SGI Altix 3700 Bx2 – 64 processors Intel Itanium 2

12/18/2008

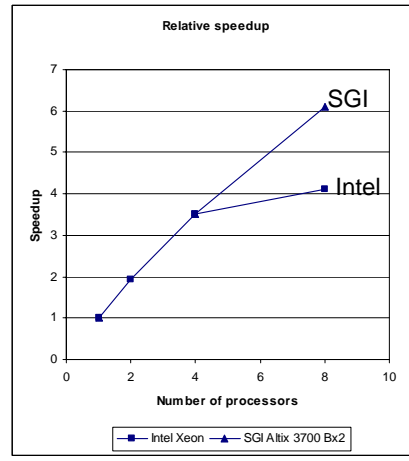
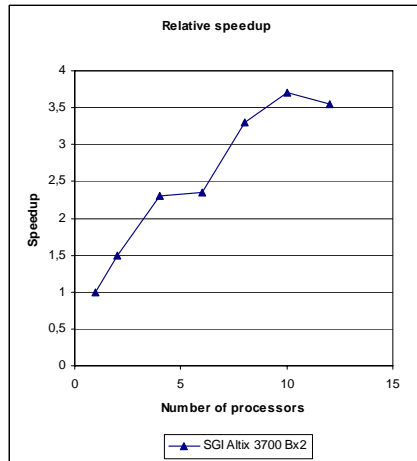
PELAB - Linköping University

30

Measurements (100000 steps Flexible Shaft Model)

Task-merging, MPI, SGI Altix

Pipelined, Pthreads, SGI, Intel Xeon



12/18/2008

PELAB - Linköping University

31

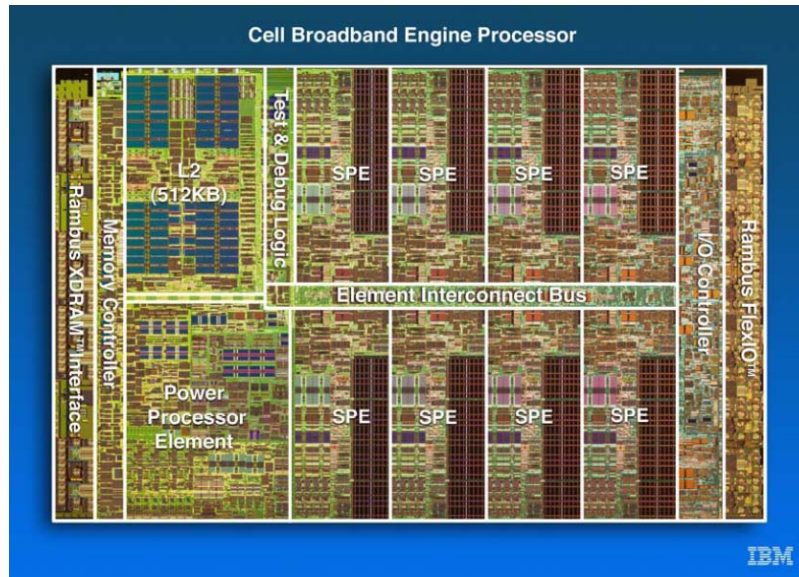
CELL BE Implementation and Measurements

12/18/2008

PELAB - Linköping University

32

CELL BE Processor



12/18/2008

PELAB - Linköping University

33

CELL BE Processor (2)

- One main 64-bit PPE processor (PowerPC)
 - Power Processor Element, 2 hardware threads
 - Good at control tasks, task switching, OS-level code
 - SIMD unit VMX
- 8 SPE processors (RISC with 128bit SIMD)
 - Synergistic Processor Element
 - Good at compute-intensive tasks
 - Small local memory 256KB (code and data)
 - No direct access to main memory – need DMA transfers
- Internal communication: Signals, Mailboxes
- Interface to Main memory (off-chip, ~ 512 MB or more)

12/18/2008

PELAB - Linköping University

34

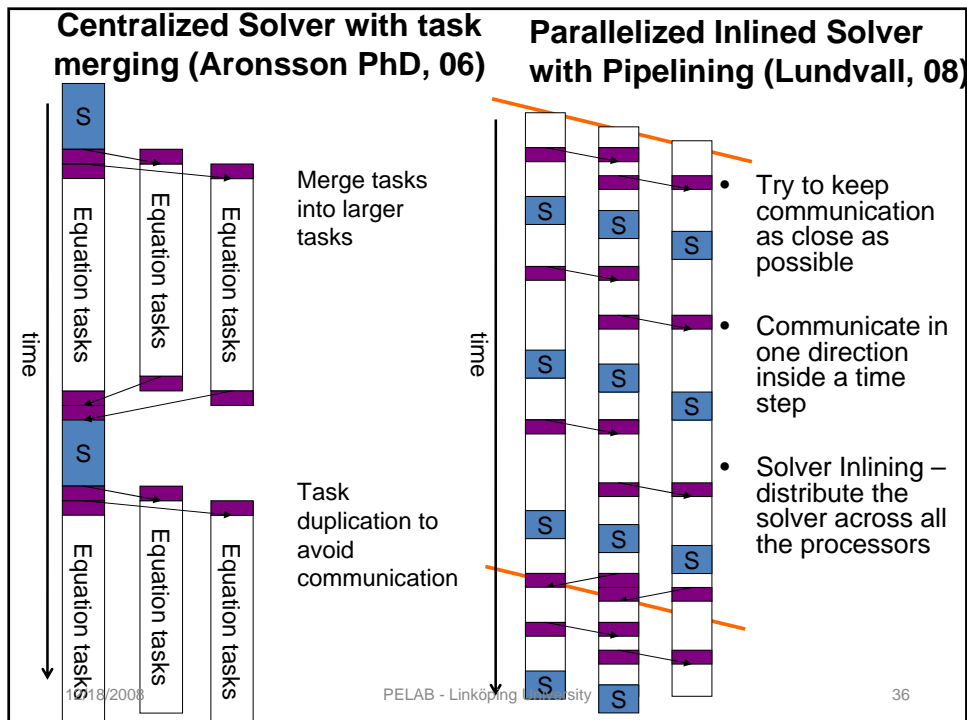
Flexible Shaft CELL Implementation

- Retargeted generated parallel code to CELL
 - Adapted (Flexible Shaft) code from OpenModelica Compiler for CELL architecture
- Future work: enhance OpenModelica compiler to produce CELL BE code
- The generated code follows the parallelization approach described earlier:
 - The **solver is inlined** in the task graph
 - The task graph spread out over several processes in a way that makes **pipelining** possible

12/18/2008

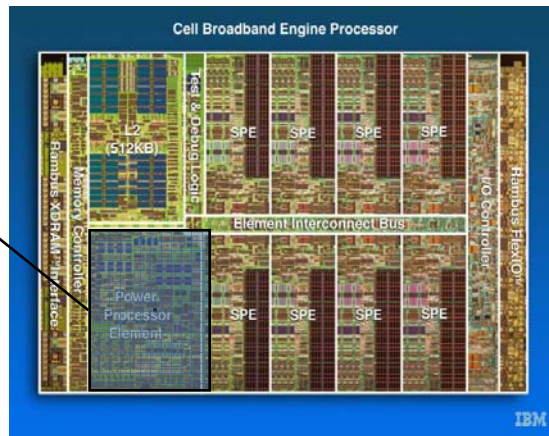
PELAB - Linköping University

35



Implementation Outline

- **Read initialization data** from file into main memory buffers
- **Create pthreads** and SPE contexts, load SPE programs, etc.
- **Start the threads**
- **Send** (using mailboxes) a pointer to a control block in main memory to the SPEs. The control block contains **pointers to main memory buffers**.



12/18/2008

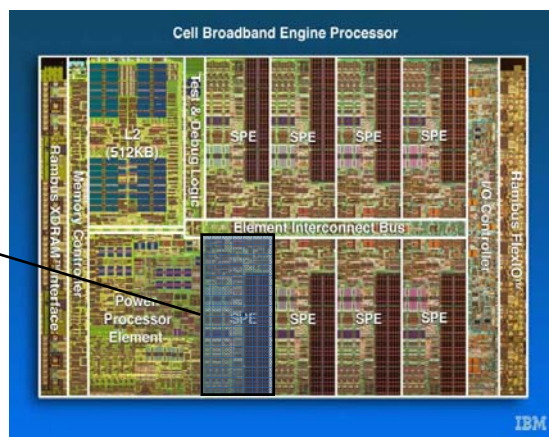
PELAB - Linköping University

37

Implementation Outline (2)

Each SPE:

- **Fetch** control block using DMA
- **Read initialization data with DMA** to local store (using the pointers in the control block)
- **Iterate local code** N steps **communicate** with neighboring **SPEs**
- **Send back** result data to **main memory** (in some steps)



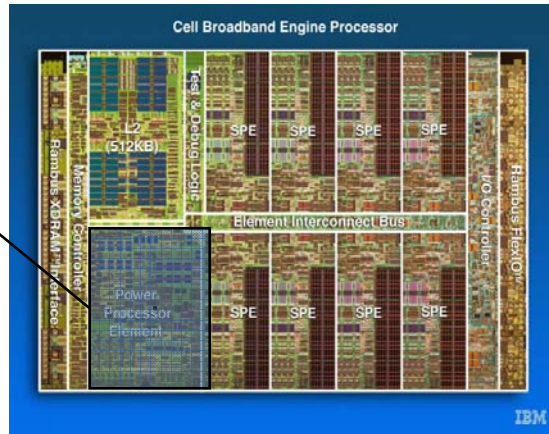
12/18/2008

PELAB - Linköping University

38

Implementation Outline (3)

- Write result data into file(s)
- Terminate program

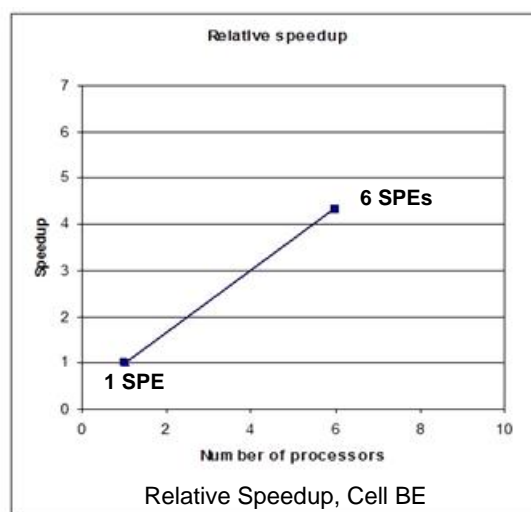


12/18/2008

PELAB - Linköping University

39

Measurements of Relative Speedup (100000 steps Flexible Shaft Model, using 6 SPEs)



12/18/2008

PELAB - Linköping University

40

Conclusions and Future Work

Conclusions and Future Work

- Good speedup on different architectures
- Test case is really too small. Larger, more complex models, may scale better
- Algebraic loops (i.e. equation SCCs)
 - Can be parallelized further using RHS task merging/scheduling (from task graph) and solver inlining (here)
- The CELL implementation is slow
 - Some of the threads spend a lot of time **waiting for DMA** to complete/synchronizing
 - We have **not utilized SIMD** Instructions yet
 - Better support for double precision calculation in next CELL version (**double precision is slow** in current CELL)
- Also try generating NVidia CUDA code

Thank you!
Questions?