Implementation and Evaluation of a PDE-solver using ParModelica

Gustaf Thorslund

Department of Computer and Information Science
Linköping University

2016-02-01

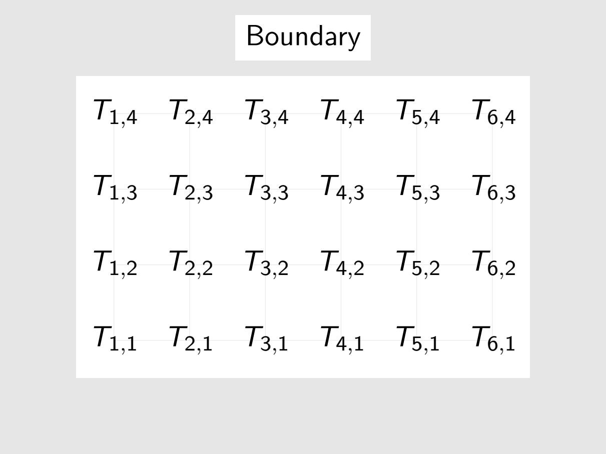
Heat Equation

$$\begin{aligned}\frac{\partial T}{\partial t} &= \kappa \nabla^2 T + \frac{\kappa h}{\lambda} \\ &= \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + \frac{\kappa h}{\lambda}\end{aligned}$$

PDE Extension

```
model HeatInPlane
  parameter Real c;
  parameter Real q;
  parameter Real h;
  field Real T(domain=omega);
equation
  c*der(T) = pder(T,D.x2) + pder(T,D.y2)
                  indomain omaga.interior;
  c*pder(T,D.x) = q+h*(T_ext-T)
                  indomain omega.left;
  T = 50          indomain omega.right;
  pder(T,D.y) = 0   indomain omega.top;
  pder(T,D.y) = 0   indomain omega.bottom;
end HeatInPlane;
```

Method of Lines



Discretised Heat Equation

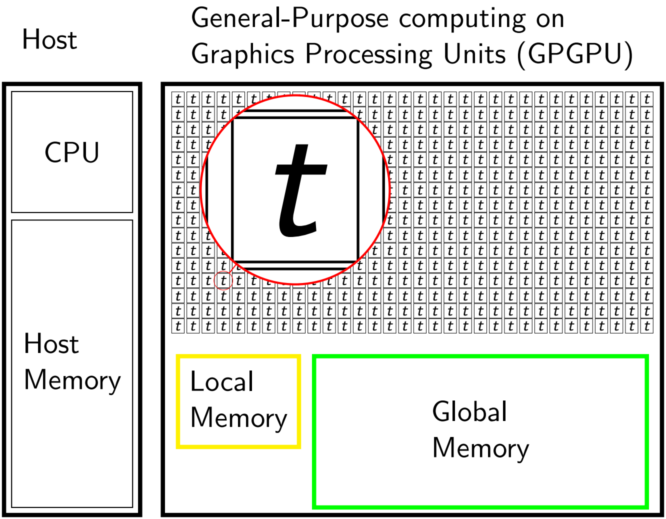
$$\frac{\partial T_{i,j}}{\partial t} = \kappa_{i,j} \nabla^2_{i,j} T_{i,j} + \left(\frac{\kappa h}{\lambda}\right)_{i,j}$$

$$= \kappa_{i,j} \left(\frac{\partial^2 T_{i,j}}{\partial x^2} + \frac{\partial^2 T_{i,j}}{\partial y^2}\right) + \left(\frac{\kappa h}{\lambda}\right)_{i,j}$$

$$\frac{\partial^2 T_{i,j}}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 T_{i,j}}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2}$$

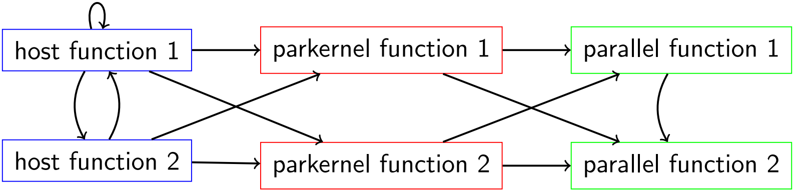
Computer equipped with GPGPU



ParModelica extensions

- parglobal/parlocal memory
- parkernel/parallel function
- parfor loop
- OpenCL as target language

ParModelica call chain



parallel function

```
parallel function pder
  input Real A[:];
  input Integer index;
  input Real h = 1;
  output Real result;
algorithm
  // ...
  if index == 0 then
    result := A[0]/h;
  else
    result := (A[index] - A[index-1])/h;
  end if;
end pder;
```

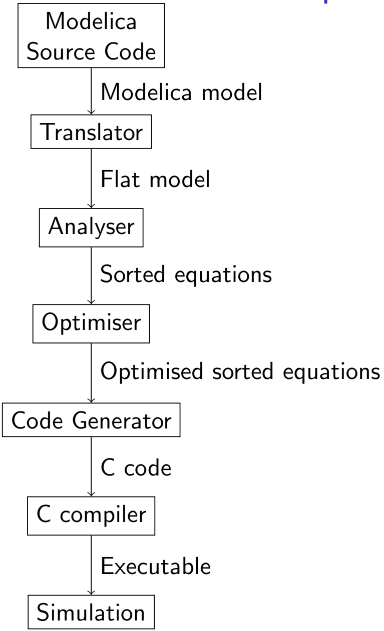
parkernel function

```
parkernel function derKernel
  parglobal input A[:];
  parglobal output B[size(A,1)];
algorithm
  for i in
    oclGetGlobalId(1):
    oclGetGlobalSize(1):
    size(A,1)
  loop
    B[i] := pder(A, i);
  end for;
end derKernel;
```

Calling a kernel function

```
function parMaxAcceleration
  input Real X[:];
  output Real maxAcceleration;
protected
            Real A(size(X,1));
  parglobal Real pX(size(X,1));
  parglobal Real pV(size(X,1));
  parglobal Real pA(size(X,1));
algorithm
  pX := X;
  pV := derKernel(pX);
  pA := derKernel(pV);
  A := pA;
  maxAcceleration := max(A);
end parMaxAcceleration;
```

OpenModelica compiler



Runge-Kutta 3(2)

$$k_1 = f(x_n, x_n)$$

$$k_2 = f(t_n + \frac{1}{2}h, x_n + \frac{1}{2}hk_1)$$

$$k_3 = f(t_n + \frac{3}{4}h, x_n + \frac{3}{4}hk_2)$$

$$x_{n+1}^{(3)} = x_n + (\frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3)h$$

$$k_4 = f(t_n + h, x_{n+1})$$

$$x_{n+1}^{(2)} = x_n + (\frac{7}{24}k_1 + \frac{1}{4}k_2 + \frac{1}{3}k_3 + \frac{1}{8}k_4)h$$

Where to implement function f?

- Ordinary Modelica function
  - Would result in too many kernel calls
  - Too slow
- parkernel function
  - Would result in too many kernel calls
  - Too slow
- parallel function
  - Can be called from parkernel function
  - Can be called from parallel function
  - Cannot synchronise between workgroups
  - Harder to create intermediate fields
  - Solver needs to synchronise calls
  - Several calls will be done at different points over the fields

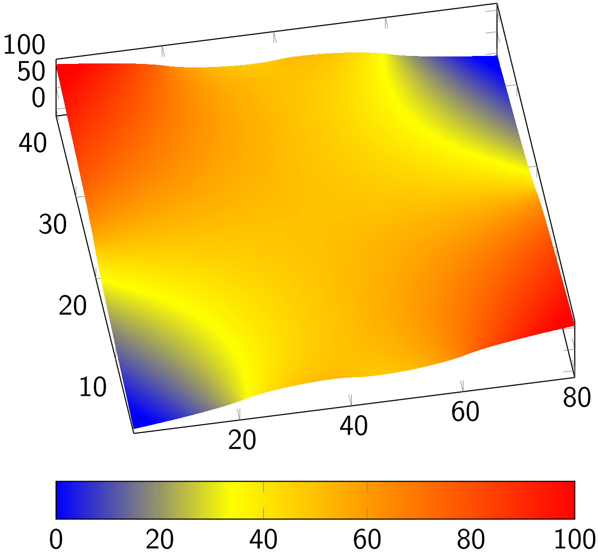
Input to Solver

```
parallel function ParDerState
  "Calculate the state derivative"
  ...
  input Types.Field[:] state
    "Array of state fields";
  input Real var[:];
  input Types.Field ext[:];
  input Real t
    "Time to calculate the state derivative at";
  input Integer i,j,k
    "Discrete coordinate within field";
  output Real value1;
  ...
```

Input to Solver cont..

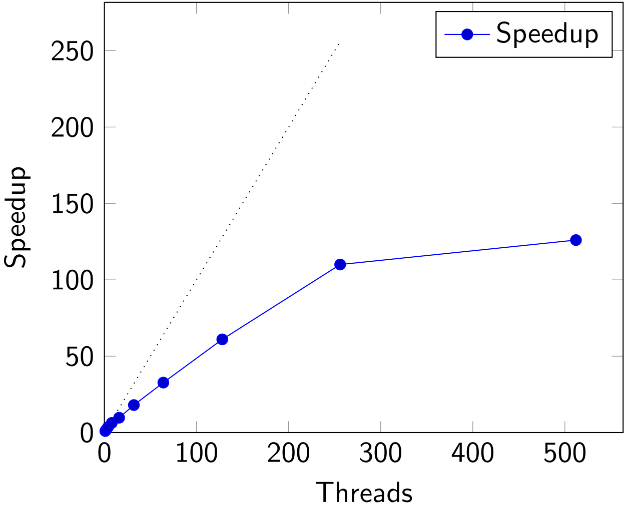
```
protected
  // User defined
  Real d2Tdx2, d2Tdy2;
  Real c = var[1];
algorithm
  // User defined
  nDer := 0; // Perfect insulation
  d2Tdx2 := Pder.Pder2Neumann(f=state, fi=1,
                              i=i, j=j, k=k,
                              dim=1, nder=nDer);
  d2Tdy2 := Pder.Pder2Neumann(f=state, fi=1,
                              i=i, j=j, k=k,
                              dim=2, nder=nDer);
  value1 := c*(d2Tdx2 + d2Tdy2)*ext[1,i,j,k];
end ParDerState;
```

Simulation



Speedup...

...when doing $\sim$ 40 timesteps per kernel call, with a total of $\sim$ 160 evaluations of the state derivative.



Conclusions

- ParModelica can be used for simulating PDEs with good speedup, in some cases
- ParModelica can be used for evaluating performance of a parallel solver
- A research project can be woken up, and enhanced
- ParModelica does have a bit of bottleneck communicating with GPU
- Absence of procedures (i.e. input/output variables or call by reference) makes abstraction harder when updating part of a matrix

Further Work

- Can overhead of ParModelica be limited by lazy copying between host/GPU? Previous master's thesis at PELAB suggest communication overhead can be a bottleneck.
- Integrate a PDE-solver into the OpenModelica compiler and simulation runtime, most probably done using C, C++, OpenMP, OpenCl, CUDA/C...
- Evaluate other solvers
- Visualisation of simulation results
- More models and evaluation of simulation result

Questions?

- Gustaf Thorslund, MSc Applied Physics and Electrical Engineering with a Software Engineering profile
- Master's thesis: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-120079
- gustaf@thorslund.org
- Current work: Senior Technical Support Engineer at Oracle, specialised in MySQL Cluster (joined MySQL AB 2007)