# CSE-633 Parallel Algorithms

Spring 2009

# Stationary Temperature Distribution In A Square Plate

Jesper Dybdahl Hede

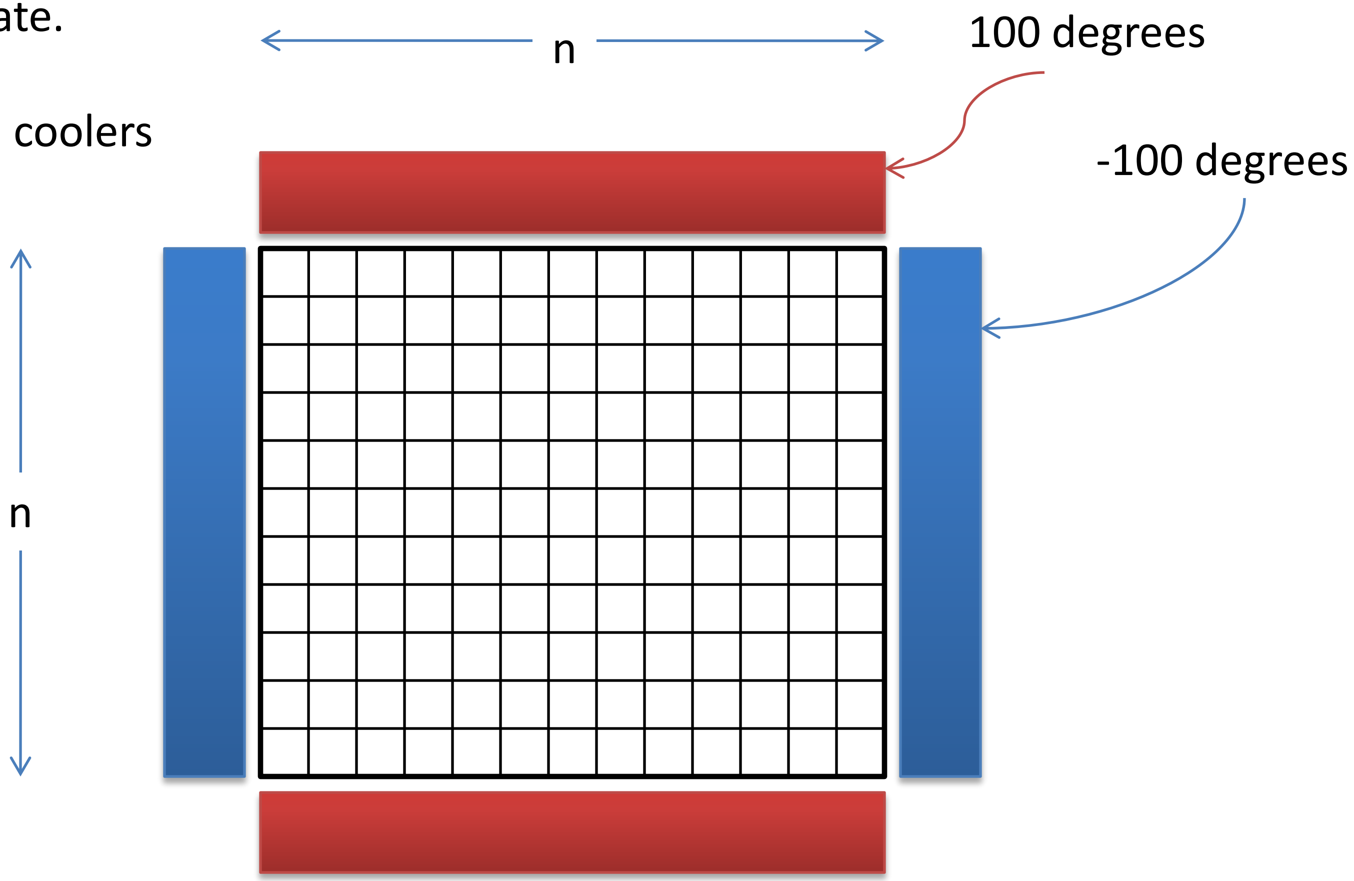
UB#: 35964041

# Problem to Solve

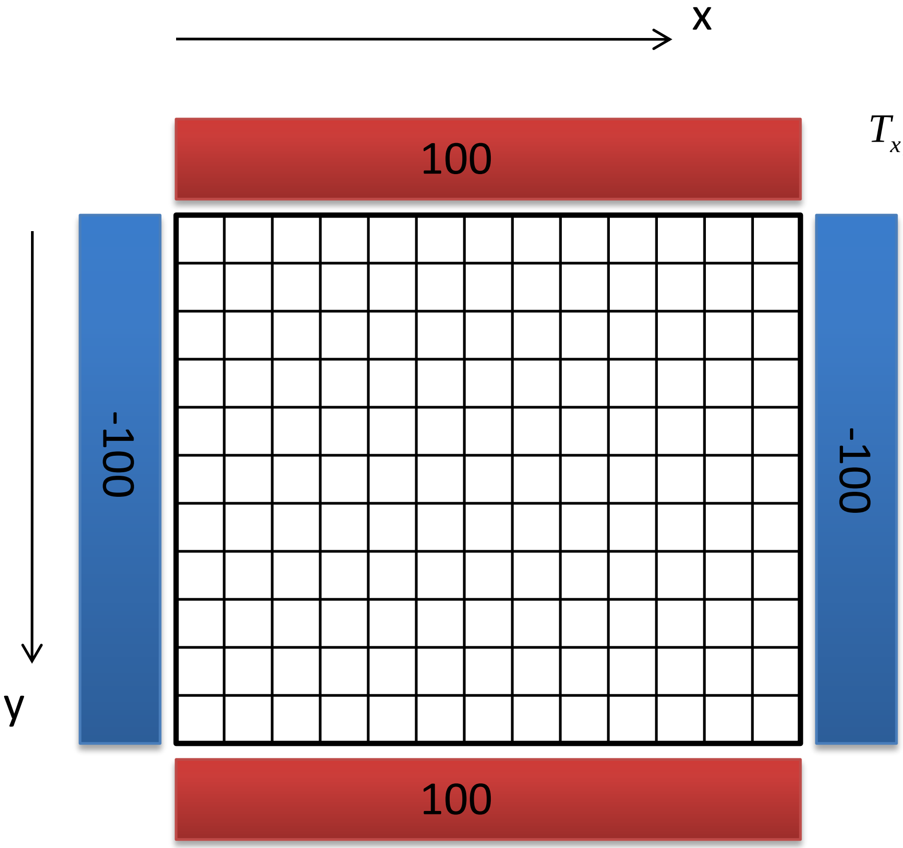
n x n size plate.

$n^2$ fields.

Heaters and coolers on edges



# Problem to Solve



Iterative process:

- In iteration i a new temperature value is calculated for each field on the plate, using the temperatures from iteration i-1:

$$T_{x,y}(i) = \frac{T_{x-1,y}(i-1) + T_{x+1,y}(i-1) + T_{x,y-1}(i-1) + T_{x,y+1}(i-1)}{4}$$

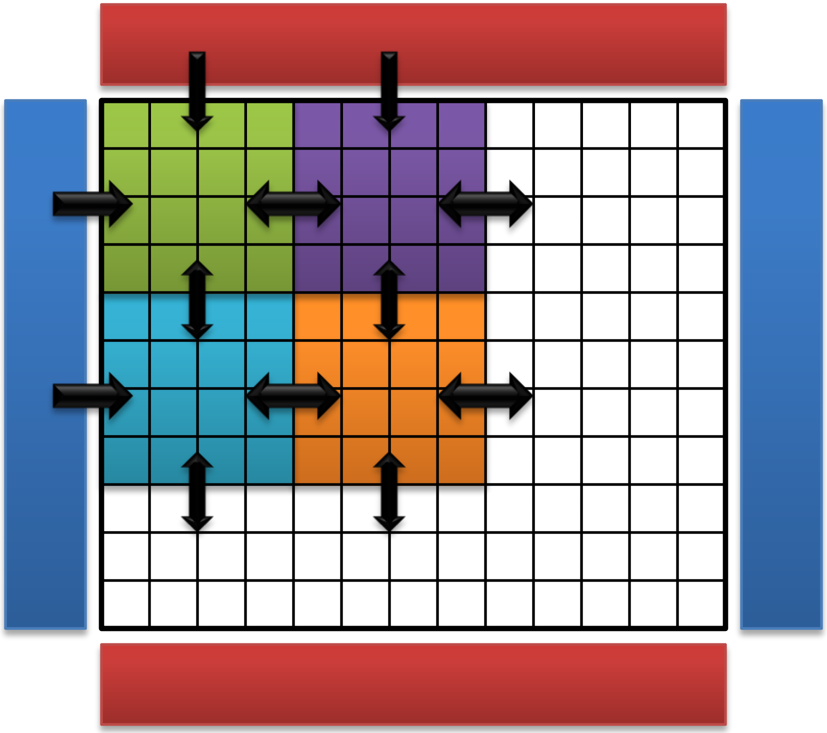
# Solving the Problem

- Several processors to solve the problem.
    - h horizontal
    - v vertical
    - h*v total processors.

- Each processors gets an equal part of the plate, i.e. a smaller rectangle.

- Each processor does all calculations within its rectangle and communicates values on edges with neighbouring processors.

- Processors on edges will use values from heaters/coolers, i.e. static values.



# Algorithm

```
Initial calculations (find neighbours/setup arrays)

Loop:

    Calculate new values for each field rectangle

    Synchronize with neighbours

Gather and save result in text file
```

Basically, the loop is being parallellized.

# Implementation

h-by-v mesh

Communication diameter: $\Theta(\max(h,v))$

Bisection width: $\Theta(\min(h,v))$

h-by-v in stead of x-by-x → More options for #PEs.

Loop stop condition:

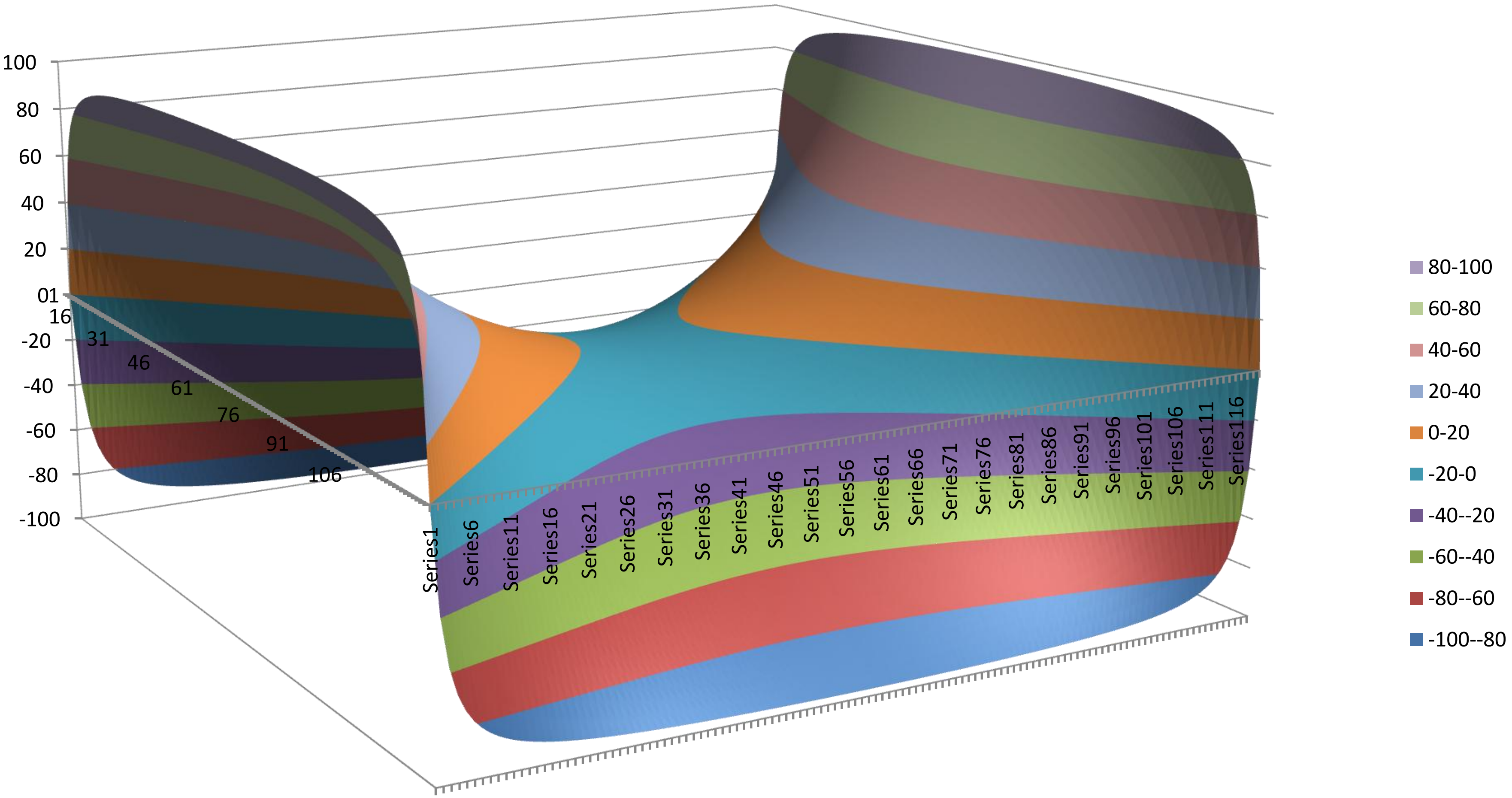
Each processors communicates an INT to neighbors along with edge values.

```
if(PE updates over threshold) Send 0
else Send min known value +1
```

min known value is min of own value and received values in last iteration.

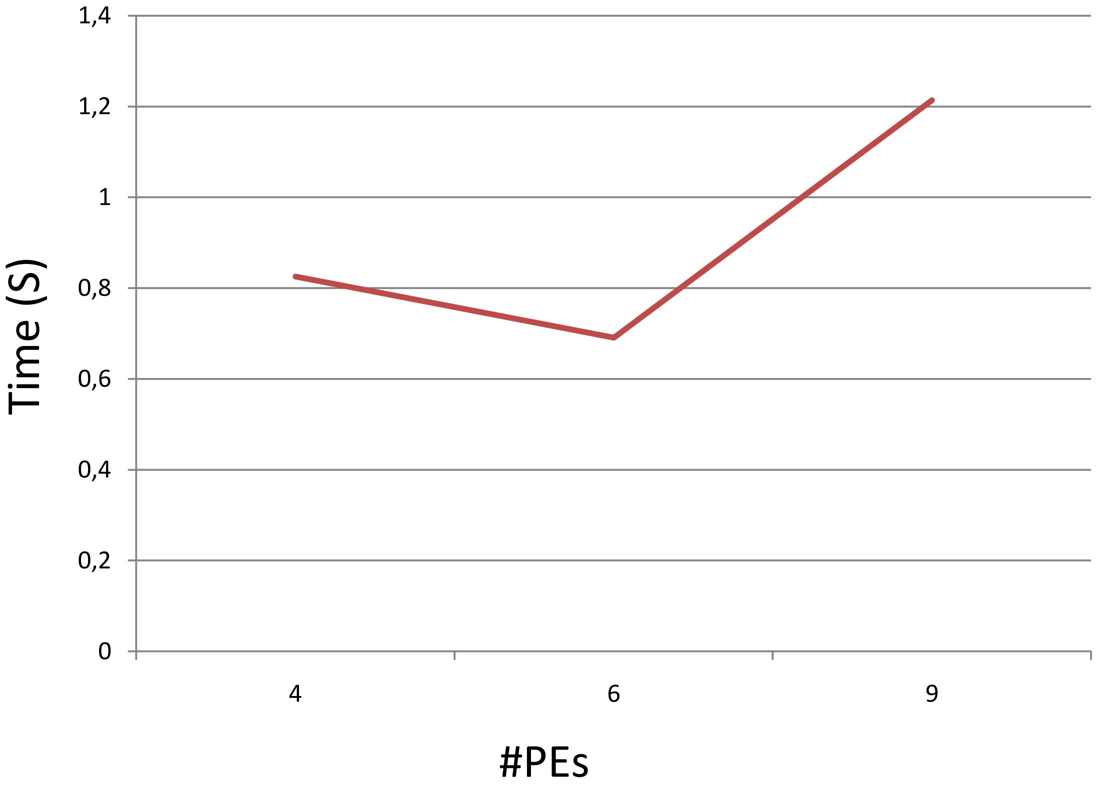
Any PE: If min known value > v+h then end loop

# Result: Temperature in plate



# Problem!

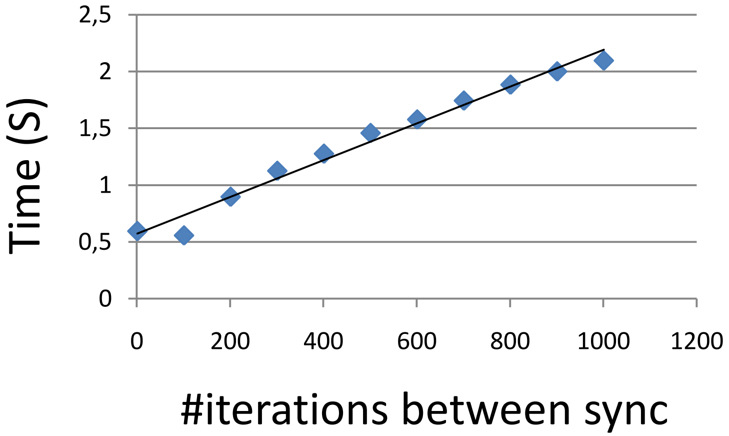
Program turns out to be less than perfect for parallelization.



# Problem!

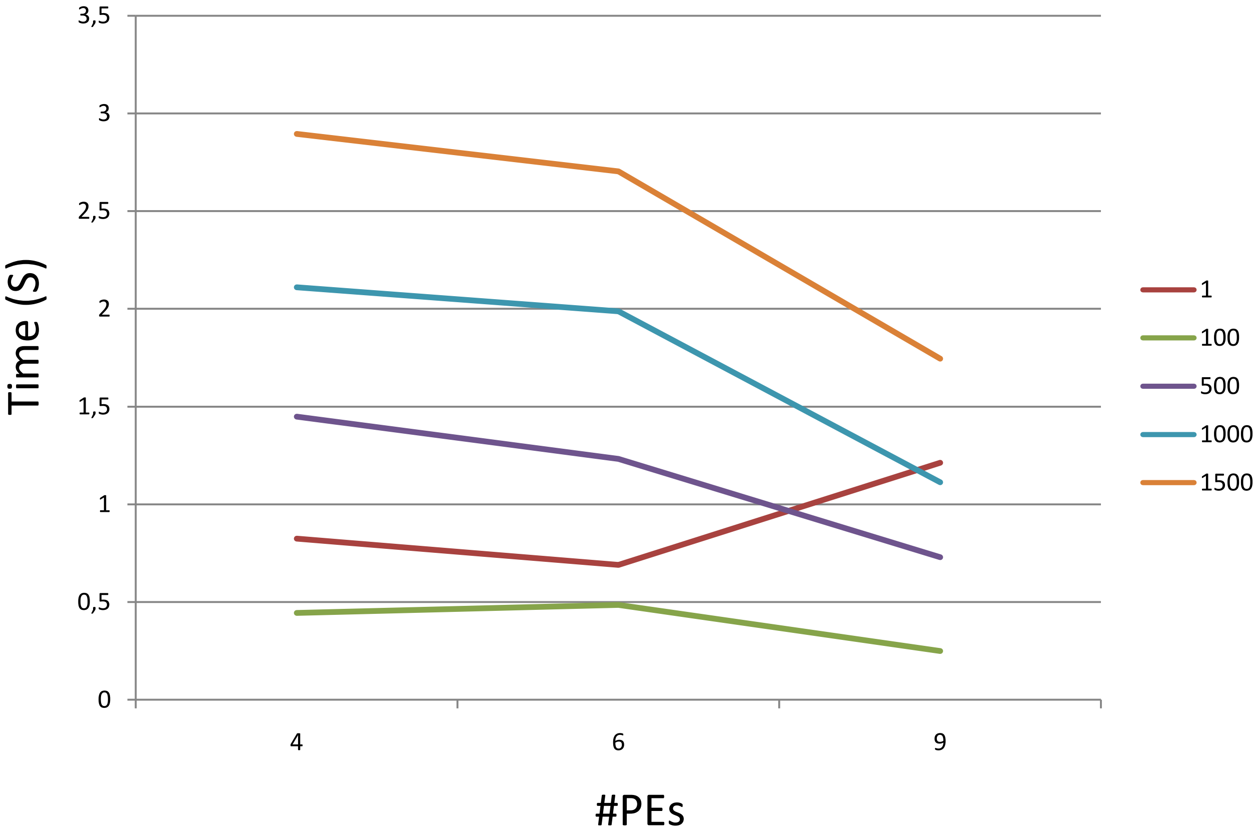
- MPI communication dominates arithmetic calculations
- Not able to make plate as large as wanted (got error above 150x150 fields)

- Solution?
  - Do more iterations of calculations between communication.
    - Reduces #sync rounds from ~2000 to ~25



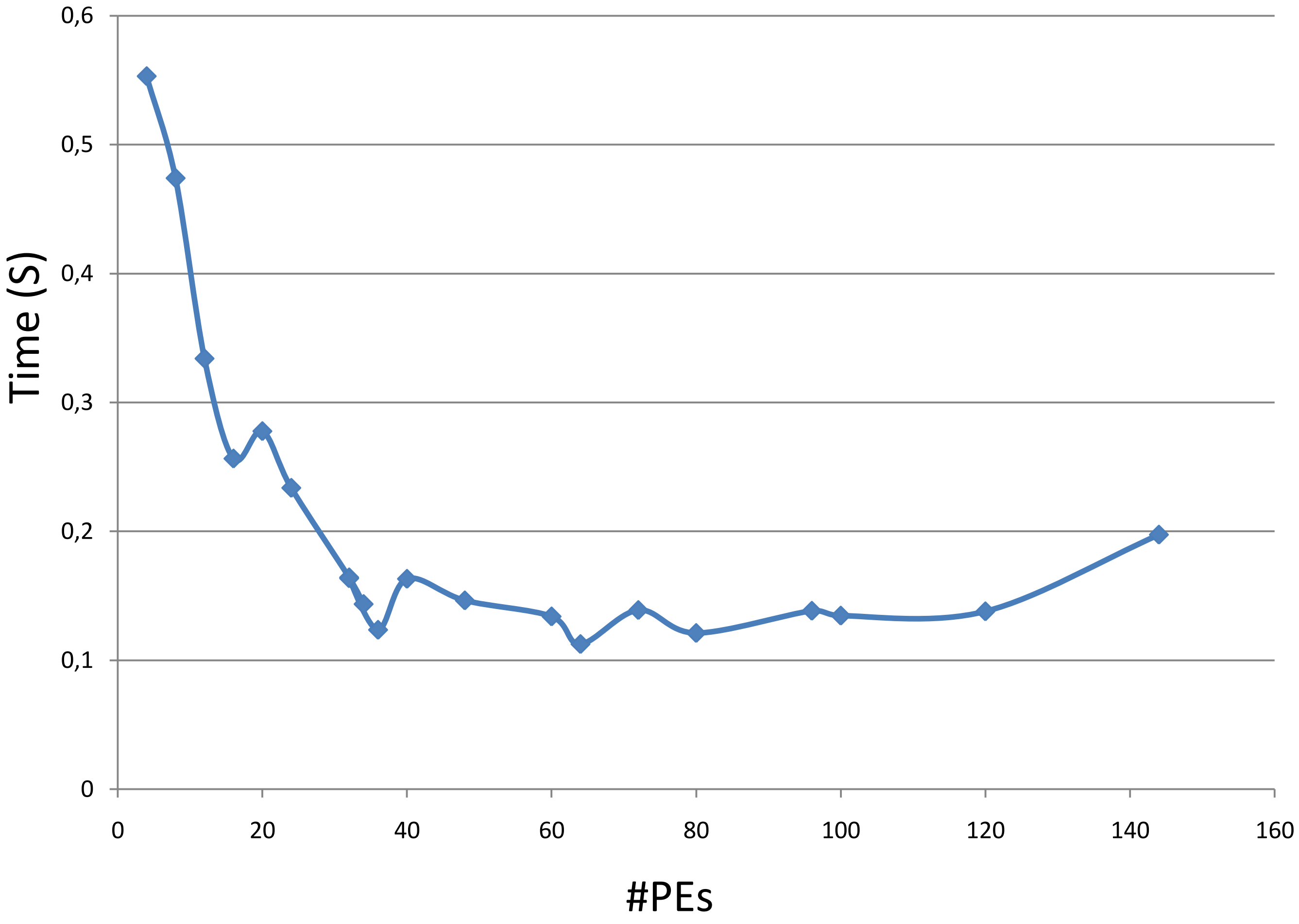
#PEs: 4

- 100 iterations per syncronization is the sweet spot!



# Results: Running time

- Runs with varying # PEs, with 100 iterations per sync. Average over 10 runs.



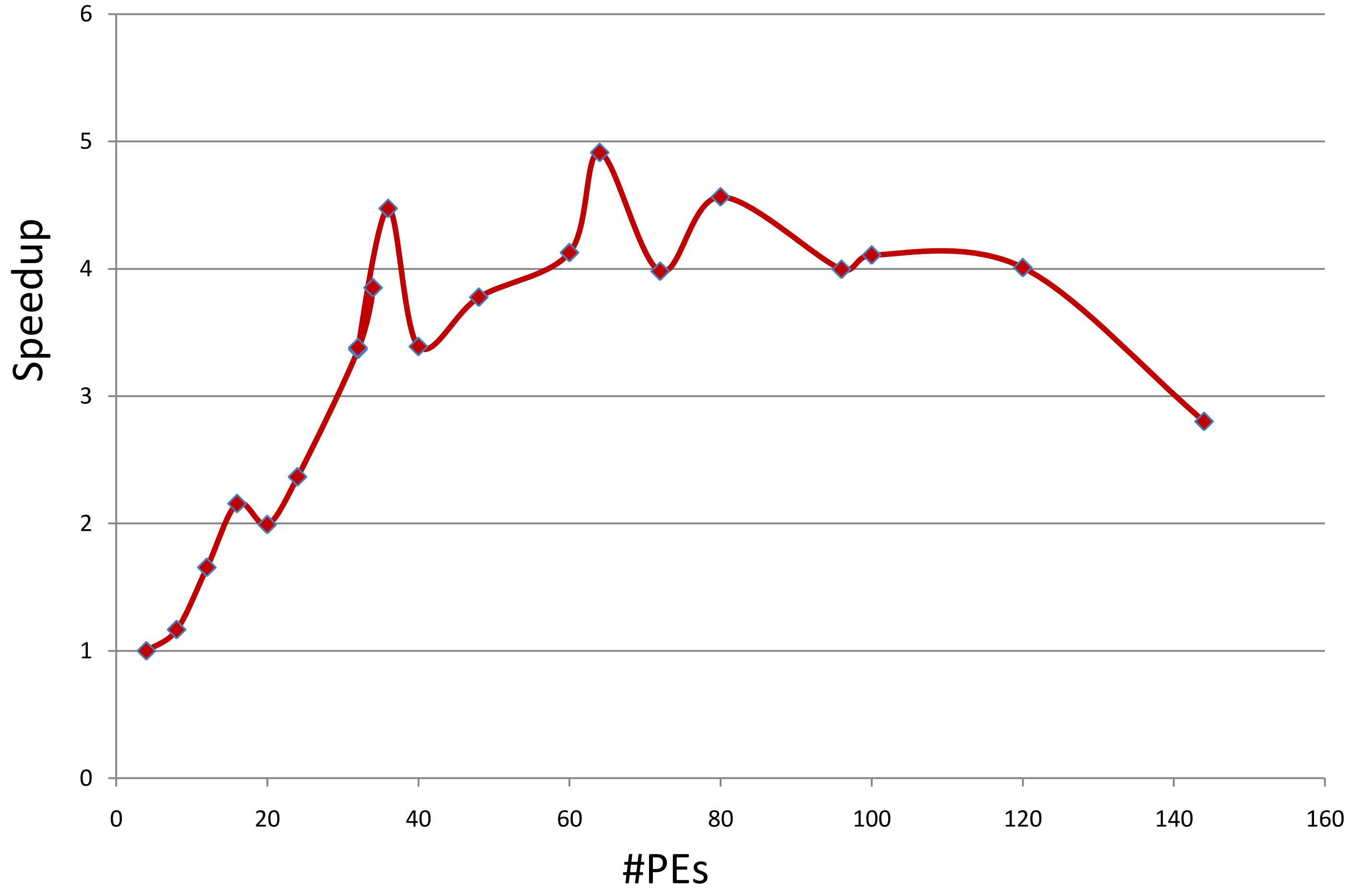
# Results: Running time

- Runs with varying # PEs, with 100 iterations per sync. Average over 10 runs.
  - Minimum at 64 PEs: 0,112559s.
  - Little gains > 30 PEs.

  - Smaller speedup than expected.

# Results: Speedup

- Runs with varying # PEs, with 100 iterations per sync. Average over 10 runs.


Speedup
#PEs