

# Applications of high level software for parallel solution of Partial Differential Equations

Åsmund Ødegård

Simula Research Laboratory AS

30. March, 2006

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI
- 7 Final words

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI
- 7 Final words

# The papers of the Thesis

- Using BSP and Python to Simplify Parallel Programming. Konrad Hinsén, Hans Petter Langtangen, Ola Skavhaug, and Åsmund Ødegård. Published in *Future Generation Computer Systems*, pp. 123-147, vol 22, 2006.
- PySE; Python Stencil Environment - Solving Partial Differential Equations with Python. Åsmund Ødegård. To be submitted.
- Fully Implicit Methods for Systems of PDEs. Åsmund Ødegård, Hans Petter Langtangen, and Aslak Tveito. Published in Langtangen and Tveito (eds): *Advanced Topics and Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*, Lecture Notes in Computational Science and Engineering, Springer-Verlag, 2003.

# The papers of the Thesis, continued

- Parallel Simulation of 3D nonlinear acoustic fields on a Linux cluster. Xing Cai and Åsmund Ødegård. Published in *Proceedings of 2nd IEEE International Conference on Cluster Computing*, pp. 185-192, 2000.
- On the Performance of PC Clusters in Solving Partial Differential Equations. Xing Cai and Åsmund Ødegård. *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, 2001.
- Finite Element Modelling of Pulsed Bessel Beams and X-Waves using Diffpack. Åsmund Ødegård, Paul D. Fox, Sverre Holm, and Aslak Tveito. *Proceedings of 25th International Acoustical Imaging Symposium*, pp. 59-64, 2000.

# Motivation

A Partial Differential Equation (PDE):

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f$$

- A large number of physical and natural processes can be modelled by PDEs.
- A PDE can usually not be solved analytically.
- A large amount of software for solving PDEs exist.
- The main objective of this thesis is to discuss new ways of implementing software for solving PDEs and systems of PDEs.

# Why do we want to use High Level Software?

- High and low are only loosely defined.
- Traditionally, low level languages are used in scientific computing for best possible efficiency.
- Coding in such languages can be complicated, tedious and error-prone.
- Scientific computing is much more than just number crunching.
- Researchers in computational science tend to prefer high level environments, like matlab, because they feel more productive!

# Other topics: Cluster computing and Ultrasound

## Performance of Clusters

During the late 90's, clusters of low-cost computers was made popular in scientific computing, through the Beowulf project.

To see whether such systems was of any use for us, we had to evaluate the performance for PDE based applications.

## Simulation of Acoustical ultrasonic waves

Simulation of ultrasound is important related to:

- Design of transducers
- Enhancement of imaging, based on wave properties.

My task was to implement flexible software for Finite Element simulations of acoustical waves.

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI
- 7 Final words

# Who did what in paper I

## *Using BSP and Python to Simplify Parallel Programming.*

- Konrad Hinsén: Author of the Scientific module, including the BSP interface. Wrote the introductory material.
- Hans Petter Langtangen: Fortran and f2py programming, ideas and supervision, writing.
- Ola Skavhaug: Model problem, Python, C and Matlab programming, experiments, writing.
- Åsmund Ødegård: Python and C programming, experiments, writing.

# Scientific computing with Python

Python is an interpreted, object oriented programming language, with a strong and dynamic typing. Python has a clean and intuitive syntax.

- Python itself is rather slow, but:
  - Extension modules implemented in C and Fortran exists, which exhibit good performance.
  - The Numerical Python module (NumPy) is the most important.
  - It is relatively easy to develop your own extensions
- Some important tools: SWIG, f2py, SIP, boost.python

# Simplified parallel programming with BSP

BSP (Bulk Synchronous Parallel) is a simplified alternative to the more mainstream MPI.

- Program alternates between computation and communication
- Each communication step involves a synchronization, which makes deadlocks impossible
- Python BSP add support for exchanging arbitrary objects.
- BSP use tow scopes; local and global. Global objects spans all available processors.

```
x = arange(0,1,0.001); v1 = sin(x); v2 = cos(x)
v1p = ParSequence(v1); v2p = ParSequence(v2)
sum = v1p + v2p
```

BSP is not a mainstream product, and rarely available.

# Mixed language approach to solving PDEs

We implement a solver for the Black–Scholes equation, that models option pricing.

- A finite difference scheme is applied to the PDE, and the Jacobi iteration is used to approximate a solution.
- We discovered that the bottleneck in the solution procedure was the tridiagonal matrix–vector product. Hence, a C extension module was created for this operation.
- For comparison, similar solvers was implemented in Matlab, C and Fortran as well.

# Conclusions of paper I

The performance of the Python solver with C extension is comparable to that of other environments:

C	Fortran	Matlab	Python	Python+C	2cpu Python
586	542	1347	3240	1072	686

- The Python BSP also yields good parallel scaling/efficiency (not shown here).
- A high level language like Python can be used for serious problems.

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II**
- 4 Presentation of paper III
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI
- 7 Final words

# Paper II

*PySE; Python Stencil Environment – Solving Partial Differential Equations with Python.*

Consider again:

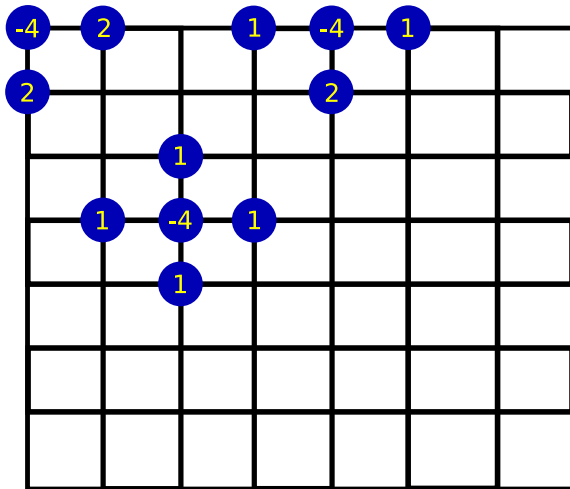
$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f, \quad x \in \Omega \in \mathbb{R}^2,$$

with appropriate boundary conditions. Centered differences for the derivative yields:

$$-\frac{1}{h^2}(u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1}) = f_{i,j}$$

PySE implemented the necessary tools to work with the Finite Difference Method (FDM) on a high level.

# A visualization of PySE



Some stencils for Laplace in 2D with Neumann boundary conditions

# Example code: Heat equation with Neumann boundary

```
from pyFDM import *
from math import *

def neumannfunc(x,y): return sin(x)*cos(y)

def initialfunc(x,y): return sin(x)*cos(y)

g = Grid(domain=([0,1],[0,1]),division=(100,100))
u = Field(g)
t = 0; dt = T/n;
A = StencilSet(g)
innerstencil = Identity(g.nsd) + dt*Laplace(g)
innerind = A.addStencil(innerstencil, g.innerPoints())
A += createNeumannBoundary(innerstencil, g, neumannfunc)
g.partition(A)
u.fill(initialfunc)
while t < T:
    u = A(u)
    t += dt
plot(u)
```

# Conclusions of paper II

- The design goals for PySE are to a large extent reached:
  - Code is clean, and looks very much like pseudo-code.
  - Parallelism is almost absent from the user-code, only a single statement is required.
  - In this case, MPI is used for parallelism, for easy deployment.
- Parallel performance is quite good for the cases we have tested.
- In absolute performance, examples show that PySE solvers can be slower than a C solver by a factor of 3.5.
- Through some examples, we show that PySE is a usable tool for doing explorative work with the finite difference method.

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III**
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI
- 7 Final words

# Who did what in Paper III

*Fully Implicit Methods for Systems of PDEs.*

- Åsmund Ødegård: Implementation, experiments, writing.
- Hans Petter Langtangen and Aslak Tveito: Ideas, model problems, proof-reading and editing.

This is the oldest work in the Thesis, mainly done in 97-98, but not published before 2003.

# The software environment Diffpack

- This work was implemented in the context of Diffpack.
- Diffpack is an object oriented library for scientific computations, implemented in C++.
- Includes a rich collection of components for implementing PDE solvers based on Finite Element or Finite Difference methods.
- Implemented from the early 90s mainly by Are Magnus Bruaset and Hans Petter Langtangen.
- In Diffpack, a solver for a PDE can be implemented as a class.

# An example

Consider the system:

$$\begin{aligned} -\nabla^2 u + u + v &= f, \\ -\nabla^2 v + u + v &= g. \end{aligned}$$

Homogeneous Neumann boundary + Galerkin Finite Elements yields:

$$\begin{bmatrix} \mathbf{A} + \mathbf{M} & \mathbf{M} \\ \mathbf{M} & \mathbf{A} + \mathbf{M} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{G} \end{bmatrix}$$

Block Jacobi ( $i = 0$ ) / Gauss-Seidel ( $i = 1$ ) approach:

$$\begin{aligned} [\mathbf{A} + \mathbf{M}] \hat{\mathbf{u}}^{(k+1)} &= -\mathbf{M} \hat{\mathbf{v}}^{(k)} + \mathbf{F}, \\ [\mathbf{A} + \mathbf{M}] \hat{\mathbf{v}}^{(k+1)} &= -\mathbf{M} \hat{\mathbf{u}}^{(k+i)} + \mathbf{G}. \end{aligned}$$

# Conclusions of paper III

- Fully implicit solvers can be implemented with flexibility similar to operator splitting based solvers.
- Coupling of already debugged and tested subsolvers together yields a development benefit.
- Through some examples, we have shown that fully implicit solvers created with our SystemFEM solution performs well.
- The framework extends straightforward to systems with any number of equations.
- Non-linear problems are more complicated, as new parts of the Jacobi matrix must be derived for a fully implicit coupling.
  - Still, we have seen that the approach also is useful for non-linear problems

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III
- 5 Presentation of paper IV and V**
- 6 Presentation of paper VI
- 7 Final words

# Who did what in paper IV and V

Both papers concerns performance of Beowulf clusters:

*Parallel Simulation of 3D Nonlinear Acoustic Fields on a Linux cluster, and*

*On the Performance of PC Clusters in Solving Partial Differential Equations.*

- Xing Cai: Navier–Stokes and water wave simulations, parallel programming, experiments, and writing.
- Åsmund Ødegård: The ultrasound application, linux clusters, experiments, and writing.

# What is a Beowulf, really

Loosely: Standard desktop computers wired together.

- Only Commodity off the Shelf (COTS) components.
- Run an open source operating system, usually Linux.
- A standard office network as interconnect.
- The main difference between a low-cost cluster and a high-end supercomputer is usually the interconnect and the memory systems.
  - In the days of our work: 100Mbps ethernet.
  - Now, 1Gbps ethernet or more.

Characteristics:

- CPUs close to “state of the art”, due to the gaming industry.
- Interconnect on real supercomputers are faster and “thighter”.

# Conclusions of paper IV and V

- For problems that results in a sparse linear system after discretization and linearization, a Beowulf cluster gives good performance
- A Beowulf gives the best price/performance ratio for “sparse” problems.
- The conventional supercomputer scales better to many processors than the Beowulf.
- The SGI Origin 2000 we compare with, scored 4.3 times better in the standard Top500 test. Thus, performance is highly application dependent.

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI**
- 7 Final words

# Who did what in paper VI

*Finite Element Modeling of Pulsed Bessel Beams and X-Waves using Diffpack.*

- Åsmund Ødegård: Finite element discretization, programming, experiments, and writing.
- Paul D. Fox: Specification of beams, model for ultrasonic fields, experiments, and writing.
- Sverre Holm: Idea, model for ultrasonic fields, proofreading and editing.
- Aslak Tveito: Idea, finite element discretization, proofreading and editing.

This work was a collaboration with the digital signal processing group at Ifi, UiO

# Challenge and Motivation of paper VI

We want to implement software for the study of limited diffraction beams in 3-dimensional ultrasonic fields.

- The simulator should handle both Bessel beams and X-Waves.
- The medium can be either homogeneous or inhomogeneous, and propagation can be either linear or nonlinear.
- In order to handle the memory requirements in the simulations, the use of parallel computers is a requirement.

# Conclusions of paper VI

- The developed simulator have the required flexibility.
- We obtain promising results for linear propagation in 3D, for small-scale transducers.
- We also obtain some preliminary result for nonlinear propagation.
- The initial results suggest strong potential for the finite element approach.
- Techniques like adaptivity needs to be investigated in order to reduce the memory demands, in order to do full-scale simulation, on e.g., the human body.

# Outline

- 1 Introduction
- 2 Presentation of paper I
- 3 Presentation of paper II
- 4 Presentation of paper III
- 5 Presentation of paper IV and V
- 6 Presentation of paper VI
- 7 Final words**

# Some final thoughts

- Computers become faster all the time, which makes it possible to solve larger and more complex problems.
- Increased complexity in problems result in more complex software.
- High level languages can be used to reduce the software complexity.
- Using Python, we have made it possible to implement parallel solvers for PDEs on a high level.
- In order to make Python a general purpose tool for explorative work in scientific computing, improvements must be done both in usability and performance.

# Last words...

Thanks goes to:

- My supervisors, Aslak Tveito and Hans Petter Langtangen
- Simula Research Laboratory AS, the SC dept. in particular.
- Institutt for Informatikk, UiO
- My dear wife - Anita
- My kids - Karoline, Rebekka, William, Henriette
- The rest of my family and all my friends.

*In the memory of Aleksander and Daniel*