Communication efficient iterative linear solvers

HPC-LEAP ESR15 Midterm Review

Teodor Nikolov

Supervisors: Prof. Andreas Frommer

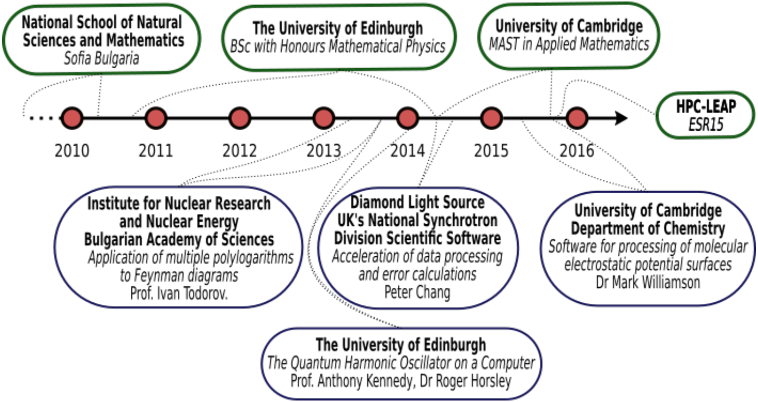
Prof. Constantia Alexandrou

Secondments supervisors: Prof. Dirk Pleiter

Jiri Kraus

April 18, 2017

Background


National School of Natural Sciences and Mathematics
Sofia Bulgaria
The University of Edinburgh
BSc with Honours Mathematical Physics
University of Cambridge
MAST in Applied Mathematics
HPC-LEAP
ESR15
2010
2011
2012
2013
2014
2015
2016
Institute for Nuclear Research and Nuclear Energy
Bulgarian Academy of Sciences
Application of multiple polylogarithms to Feynman diagrams
Prof. Ivan Todorov.
Diamond Light Source
UK's National Synchrotron
Division Scientific Software
Acceleration of data processing and error calculations
Peter Chang
University of Cambridge
Department of Chemistry
Software for processing of molecular electrostatic potential surfaces
Dr Mark Williamson
The University of Edinburgh
The Quantum Harmonic Oscillator on a Computer
Prof. Anthony Kennedy, Dr Roger Horsley

Training within the network

What I've learned

Languages and libraries: C++11/14 / C, Python, MPI 3.0, CUDA 8.0, OpenMP, Trilinos, ...

Techniques: template programming, compile-time evaluations(constexpr), vectorization, data alignment, cache-blocking, ...

Tools: CMake, Score-P, Vampir, Allinea Map, Valgrind, ...

GPU Hackathon EuroHack 2017

Team: "HPC-LEAP"

Mentors: Peter Steinbach, Anne Severt

Participants: Viacheslav Bolnykh, Srijit Paul, Guillaume Tauzin and I

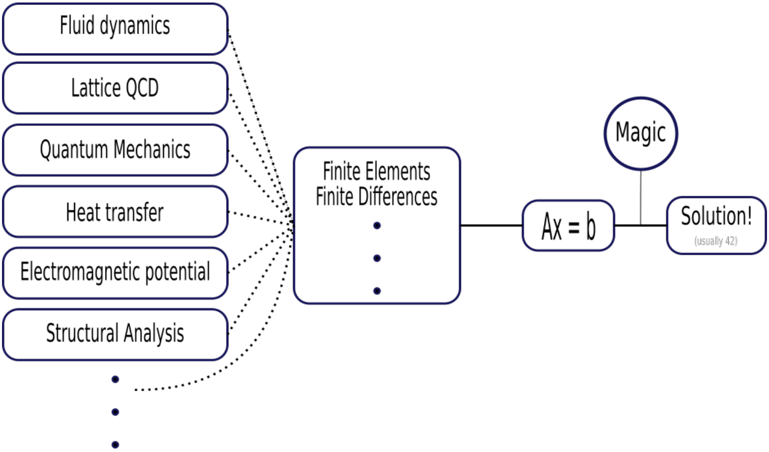
Goal: Port "Asynchronator" to NVIDIA GPUs with CUDA 8.0.

Secondments at JSC and NVIDIA

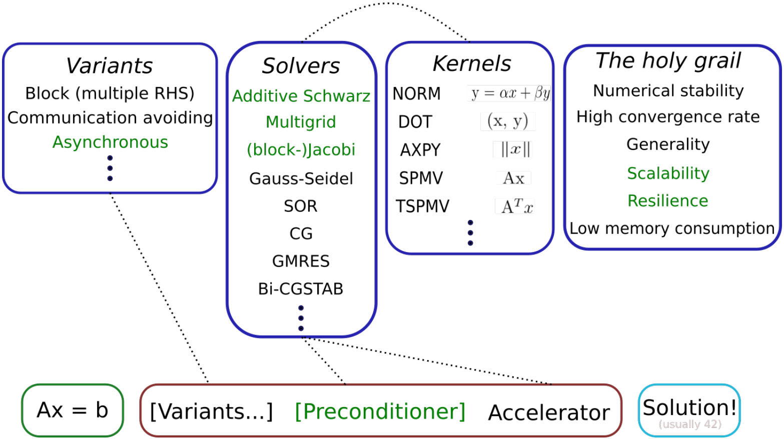
Goals: Profiling, execution analysis and GPU acceleration of "Asynchronator".

Learning something new every day thanks to: Dirk Pleiter, Salem El-Sayed Mohamed, Jiri Kraus, Andreas Herten and Markus Götz.

Large sparse linear systems



Iterative solvers



Project goals

Initial

- ☑ New asynchronous implementation strategy
- ☑ General asynchronous stencil operator
- ☑ Prototype asynchronous block-Jacobi preconditioner (D2.3) (M18)
- ☐ Multigrid preconditioner based on asynchronous smoother (D2.4) (M30)
- ☐ Convergence behaviour of asynchronous multilevel methods (D2.5) (M48)

Additional

- ☑ New asynchronous execution model: **Separate Synchronous**!!
- ☐ Mathematical properties of separate synchronous methods

Asynchronous methods

An iteration method is characterized by a sequence of operators $(T^k)_{k\in\mathbb{N}}$ and a sequence of iterates $(x^k)_{k\in\mathbb{N}}$. At each iteration $k \in \mathbb{N}$, a process updates a selected set of components $I^k \subseteq \{1, \cdots, n\}$ using all needed data without making sure it is the most recent(without synchronizing with other processes)! For $k \in \mathbb{N}$ update components

$$x_i^{k+1} = \begin{cases} x_i^k & \text{for} \quad i \notin I^k \\ (T^k \hat{x}^k)_i & \text{for} \quad i \in I^k \end{cases}$$

where

$$\hat{x}^k = (x_1^{s_1(k)}, x_2^{s_2(k)}, \cdots, x_n^{s_n(k)})$$

with

$$(s_1(k), s_2(k), \cdots, s_n(k)) \in \mathbb{N}_0^n$$

representing the iteration at which each component was last updated.

Special cases of asynchronous methods

Synchronous methods
Synchronous methods are a special case of asynchronous where the most recent data from previous iterations is available at all processes:

$$s_1(k) = s_2(k) = \cdots = s_n(k) = k$$

The meaning of k is changed: increments when all processes finish updating.

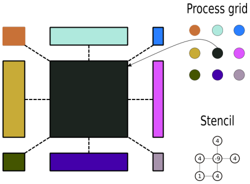
Separate synchronous methods
An idea inspired by the new asynchronous methods implementation: separately synchronize iterations and communications while avoiding race conditions! For each process p updating coordinates I^p and reading halos from iteration $l(k) \leq k$:

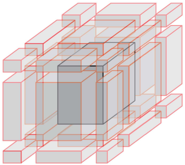
$$s_i(k) = \begin{cases} k & \text{for} \quad i \in I^p \\ l(k) & \text{for} \quad i \notin I^p \end{cases}$$

Note: All halos come from the same previous iteration!

Asynchronator [1]

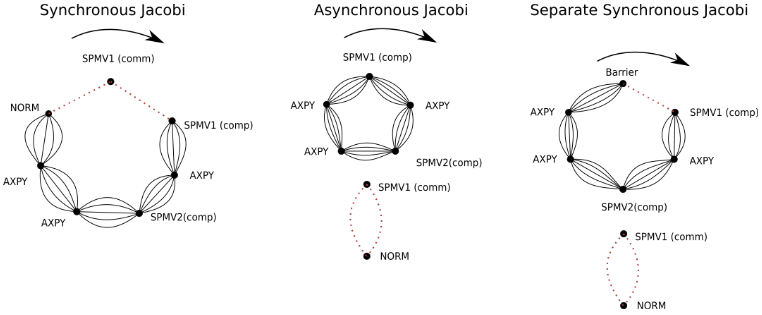
- Four Jacobi variants:
  - Synchronous
  - Asynchronous (Concurrent)
  - Consistent read asynchronous
  - **Separate synchronous**
- General distributed asynchronous stencil application
- Languages and libraries: C++11 (std::thread, templates, constexpr, smart pointers, lambdas ...) & MPI 3.0 (RMA) & CUDA 8
- Analysis tools: Vampir, Score-P, Allinea MAP, Valgrind
- Supercomputers: JURECA, JURON and JULIA at the JSC





[1] https://gitlab.com/teoHPC/asynchronator

Asynchronator Execution Model



Summary and future goals

What has been done:

- New asynchronous model!
- New asynchronous implementation strategy!
- First distributed general stencil asynchronous operator
- First asynchronous hybrid code

What is left to do:

- Wrap up the GPU implementation: the first GPU+MPI asynchronous implementation
- Integrate Asynchronator into Trilinos: use of Multilevel and block-Jacobi as preconditioners
- Convergence analysis of separate synchronous methods
- Apply new preconditioners in Lattice QCD and Fluid Dynamics