UPPSALA
UNIVERSITET

PARALUTION - a Library for Iterative Sparse Methods on CPU and GPU

Dimitar Lukarski

Division of Scientific Computing
Department of Information Technology
Uppsala Programming for Multicore Architectures Research Center (UPMARC)
Uppsala University, Sweden

GPU-enabled Libraries Course Sept, 2013

UPPSALA UNIVERSITET

Outline

Introduction and Motivation

PARALUTION Library

Solvers/Preconditioners

Integration with Application

Conclusion

UPPSALA
UNIVERSITET

Introduction and Motivation

Multi/many-core systems are here

- CPUs, GPUs, Accelerators
- Trends - **more cores** CPU(16), MIC(60), GPU(2496)
- High performance, better performance/watt ratio

Scientific computing is expected to be

- **Fast** - use this new technology
- **Scalable** - use many cores (parallel)
- **Sustainable** over years

What is the Problem?

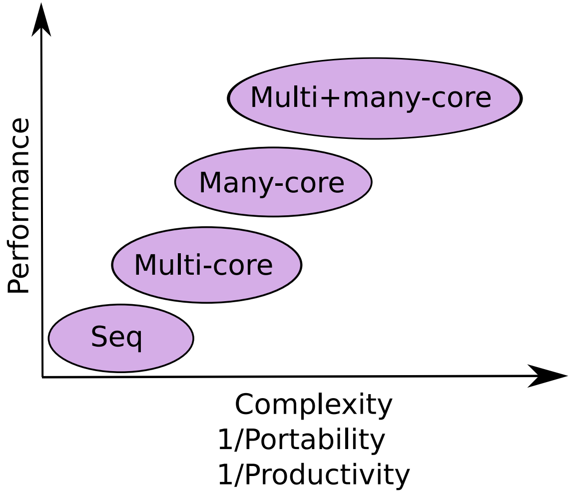
New algorithms

- High degree of parallelism
- Fine-grained parallelism
- Scalable = almost no communication

Software

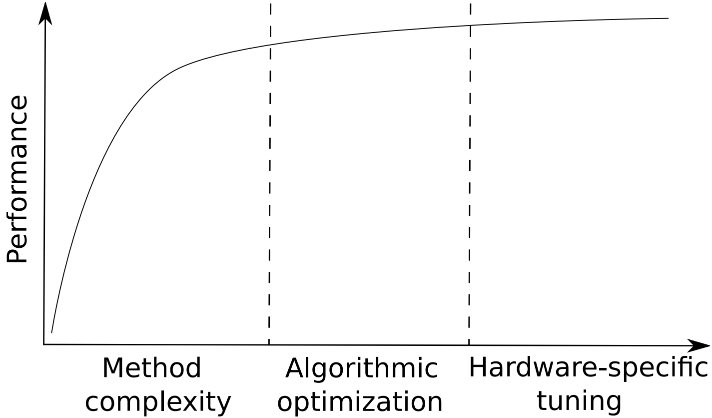
- New programming models/languages
- Hardware-specific optimization techniques

Good Parallel Programming is HARD



UPPSALA UNIVERSITET

Methods are the MOST Important



UPPSALA UNIVERSITET

Goal

Create a library for iterative methods (linear and non-linear systems)!

Math

- Linear operators (sparse/dense matrices, stencils)
- Vector routines
- Extendable

Software

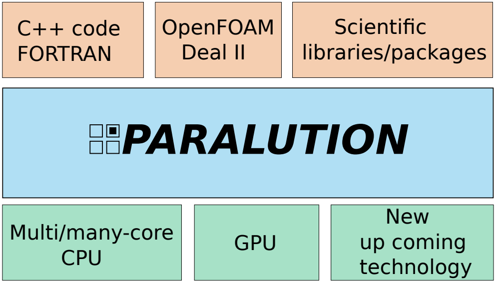
- Current multi/many-core dev, **new hardware**
- Portable code

PARALUTION

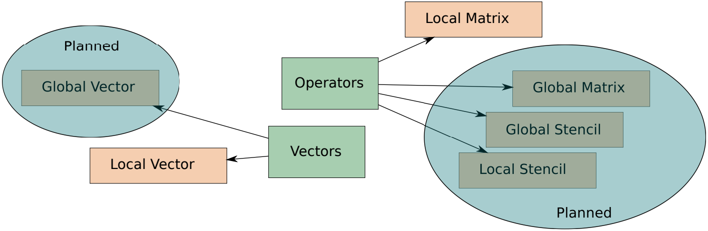
- Library for iterative sparse methods
- Targeting devices: CPUs + Accelerators (GPUs,...)
- Hardware abstraction
- No knowledge in OpenMP/CUDA/OpenCL required
- No special library/hardware required
- Portable results and code
- Run time type identification (RTTI)
- Easy to use
- Open source / GPL v3

UPPSALA UNIVERSITET

Middle-ware



Operators and Vectors



Local Vectors

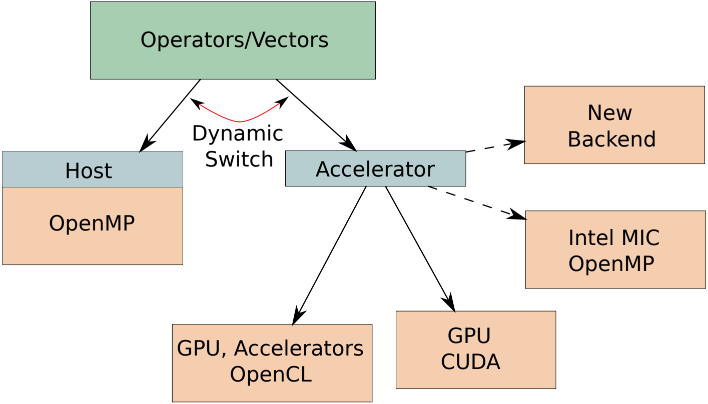
- Init, Clear
- Standard vector routines
  - Dot product
  - Vector updates
  - Norm
  - ...
- Permutations
- Copy (sub-)vector to (sub-)vector
- I/O file
- Raw access via pointers

Local Matrices

- Init, Clear
- Formats – CSR, MCSR, COO, ELL, DIA, HYB, DENSE
- Conversion
- Matrix-vector multiplication
- Permutation
- Extract a sub-matrix
- I/O file
- Graph analyzer (CPU)
  - Multi-coloring
  - Maximal independent set
- Factorization
- Some wrapper to Intel/MKL

UPPSALA
UNIVERSITET

Operators and Vectors



UPPSALA UNIVERSITET

Source Example

```
LocalMatrix<ValueType> A;
LocalVector<ValueType> x, y ;

A.ReadFileMTX("my_matrix.mtx");
x.Allocate("vector1", mat.get_nrow());
y.Allocate("vector2", mat.get_ncol());

// y = A x
A.Apply(x, &y);

// Print the dot product of x and y
std::cout << x.Dot(y) << std::endl;
```

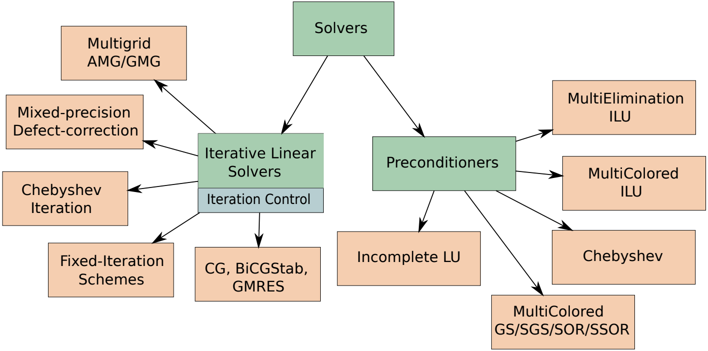
Source Example

```
LocalMatrix<ValueType> A;
LocalVector<ValueType> x, y ;

A.ReadFileMTX("my_matrix.mtx");
x.Allocate("vector1", mat.get_nrow());
y.Allocate("vector2", mat.get_ncol());

A.MoveToAccelerator();
x.MoveToAccelerator();
y.MoveToAccelerator();

A.Apply(x, &y);
std::cout << x.Dot(y) << std::endl;
```

Initialization/Shutdown

```
#include <paralution.hpp>

using namespace paralution;

int main(int argc, char* argv[]) {

  init_paralution();

  info_paralution();

  // ...

  stop_paralution();

}
```

Solvers


Solvers
Multigrid
AMG/GMG
Mixed-precision
Defect-correction
Iterative Linear
Solvers
Iteration Control
Chebyshev
Iteration
Fixed-Iteration
Schemes
CG, BiCGStab,
GMRES
Preconditioners
MultiElimination
ILU
MultiColored
ILU
Incomplete LU
Chebyshev
MultiColored
GS/SGS/SOR/SSOR

CG Solver

```
CG<LocalMatrix<ValueType>, LocalVector<ValueType>,
ValueType > ls;

ls.SetOperator(mat);

ls.Build();

ls.Solve(rhs, &x);
```

CG Solver

```
....

ls.MoveToAccelerator();
mat.MoveToAccelerator();
rhs.MoveToAccelerator();
x.MoveToAccelerator();

ls.Solve(rhs, &x);
```

PCG Solver

```
CG<LocalMatrix<ValueType>, LocalVector<ValueType>
ValueType > ls;
MultiColoredILU<LocalMatrix<ValueType>,
LocalVector<ValueType>, ValueType > p;

ls.SetOperator(mat);
ls.SetPreconditioner(p);

ls.Build();

ls.Solve(rhs, &x);
```

PCG Solver (same procedure)


```
....

ls.MoveToAccelerator();
mat.MoveToAccelerator();
rhs.MoveToAccelerator();
x.MoveToAccelerator();

ls.Solve(rhs, &x);
```

Design and Concepts

Hardware decision

- At run time
- No template parameter

MoveToHost/Accelerator

- All objects (matrices, vectors, solvers, preconditioners, etc)
- Always a CPU implementation

Template

- ValueType - float, double, int
- Solvers - Operator/Vector type

Design and Concepts

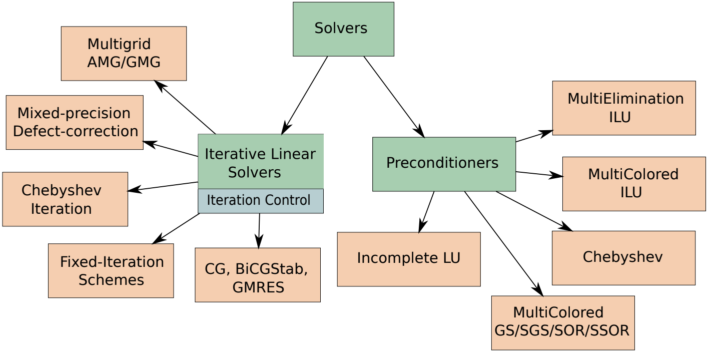
Functionality on the Accelerators

- Not all algorithms can be performed on the accelerator
- Ex: greedy multi-coloring, maximal independent set

The library has internal mechanism to check if a routine can be performed on the accelerator or not. If not the object is moved to the host and the routine is performed there.

- Always a CPU implementation
- Warning is printed if the object needs to be moved for the routine
- 100% portability of the code!

Iterative Linear Solvers



- Preconditioner = a solver
- Flexible design

Solver as Preconditioner

- Every solver can be used as a preconditioner
- Set a lower stopping criteria
- or/and Fixed number of iterations

Mostly used

- GS, SOR or SGS, SSOR
- GMG/AMG

PCG-AMG

```
CG<LocalMatrix<ValueType>,
LocalVector<ValueType>, ValueType > ls;

AMG<LocalMatrix<ValueType>,
LocalVector<ValueType>, ValueType > p;

p.InitMaxIter(2);

ls.SetPreconditioner(p);
ls.SetOperator(mat);

ls.Build();
```

Geometric Multigrid

Your application constructs

- Prolongation/Restriction operators or neighbor mapping vector

Pass it to PARALUTION and obtain

- Geometric Multigrid
- Interally constructed smoothers
- You can construct or pass the system matrices on all levels

Introduction and Motivation

PARALUTION Library

Solvers/Preconditioners

Integration with Application

Conclusion

Plug-ins

Deal.II

- Export/Import vector/matrices
- Call PARALUTION solvers

OpenFOAM

- Compile PARALUTION with your application
- Configure the linear solver (txt file)

FORTRAN

- Pass matrix, vector
- Select a solver
- Obtain the solution vector

Advanced Integration

- Easy integration for any C++ code
- Can be hacked for everything (e.g. FORTRAN)

Avoid init times

- Do not init PARALUTION every time you call a solver
- Move the init in the beginning of your application

Raw data via pointers

- Create PARALUTION object via data pointers
- Obtain data via pointers from PARALUTION objs
- Host (OpenMP) and GPU (CUDA) backends
- No extra data copying

PARALUTION

Library for iterative sparse methods

- Various iterative solvers
- Various preconditioners

Backends

- OpenMP, CUDA, OpenCL
- Open for new hardware
- Portable results and code

General

- No special library/hardware required
- Easy to use
- Open source / GPL v3

PARALUTION - Doc

User Manual

- Installation
- Design
- Usage
- Examples

Doxygen

- Function descriptions
- Structure of the classes

Thank You for Your Attention

PARALUTION

www.paralution.com