Metadata for Component Optimisation

Olav Beckmann, Paul H J Kelly and John Darlington

ob3@doc.ic.ac.uk

Department of Computing, Imperial College London

180 Queen's Gate, London SW7 2BZ

United Kingdom

Componentisation vs. Optimisation

- Componentisation is arguably all about abstraction and separation of concerns
  - Separating behaviour from implementation
  - Components are separately built, tested, verified . . .
- Optimisation is arguably all about adapting a component to its context of use
  - Software performance optimisation is often a disruptive, poorly separated, cross-cutting concern
- This talk describes some ideas on the role of metadata in managing the complexity of optimisation.
  - Metadata is declarative information about components and data. The aim in designing metadata has to be to manage the complexity of optimisation decisions.
  - Optimisation should be a separate concern, which does not interfere with the component code base.

Context

Context consists of

- Platform
  - architecture
  - resource availability
- Compositional structure
  - sequential composition
  - parallel composition
  - data flow
- Data
  - properties of the data that components operate on
  - layout of the data
  - aliasing information

Context is Staged

(becomes known in stages)

- Compile time
- Link time
- Deployment time
- Run-time

Context is Domain-Specific

- components
- data

Main Entities Involved in Optimisation

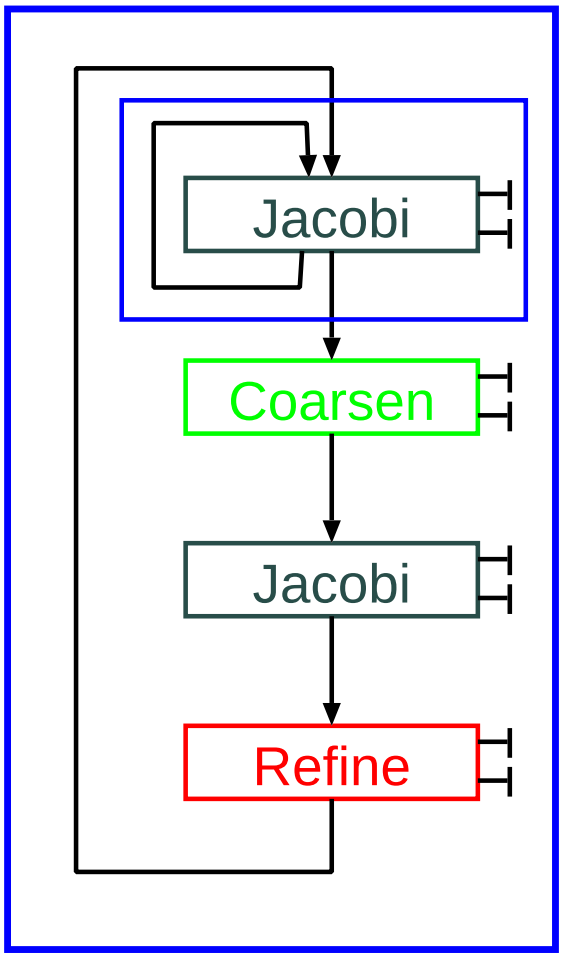
1. Components
   - Carry declarative metadata, which form the basis of optimisation decisions
   - May have a *performance interface* of tunable parameters that affect performance but not behaviour
2. Component Manager
   - Holds the workflow specified by the application “programmer”.
   - Plug-in interface for optimisations
3. Optimisations (“Meta-Components”)
   - Separate pieces of code
   - Independently developed, independently re-usable etc.
   - Work on components and/or the workflow
   - Select an efficient overall implementation

Component Metadata for Optimisation

- Component metadata for optimisation has to be extensible, with an open interface for optimisations to access (and write new) metadata.
- Constraints are used to determine the validity of applying optimisations
- Tunable Performance Parameters. Optimisation components need to select the right “settings” when components are composed
- Performance Information. Describes performance characteristics as a function of tuning parameters. *May* be dynamic, *i.e.* gathered as the component runs.

Case Study 1: Multigrid Solver

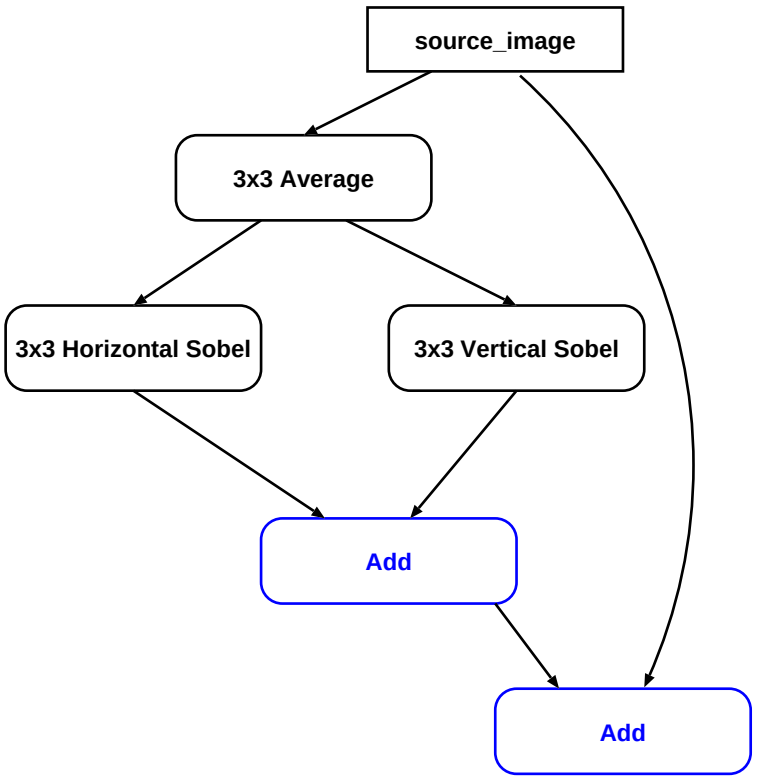
- Components involved: Relaxation (*e.g.* Jacobi), Coarsen, Refine
- Hierarchical, nested composition



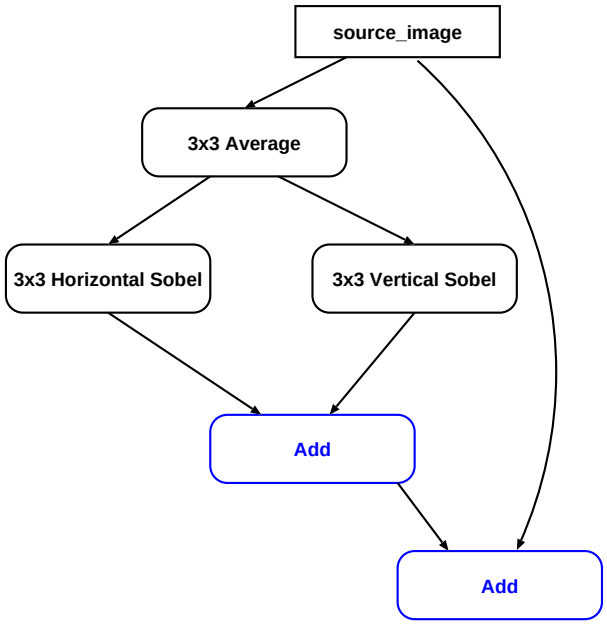
- Metadata
  - use / def sets (not enough!)
  - dependence vectors
  - performance in MFLOP/s
- Optimisations
  - Skewing
    changes dependence vectors
  - Execution by slices
    should change performance
  - (SMP) Parallelisation?

Case Study 2: Image / Movie Processing Pipeline

- For feature extraction etc, compose routines to build a data-processing network.
- This might be done dynamically in an interactive environment.

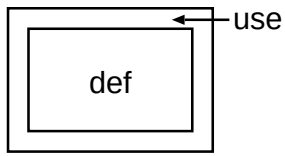


Case Study 2: Image / Movie Processing Pipeline



Metadata

- Use / Def Regions



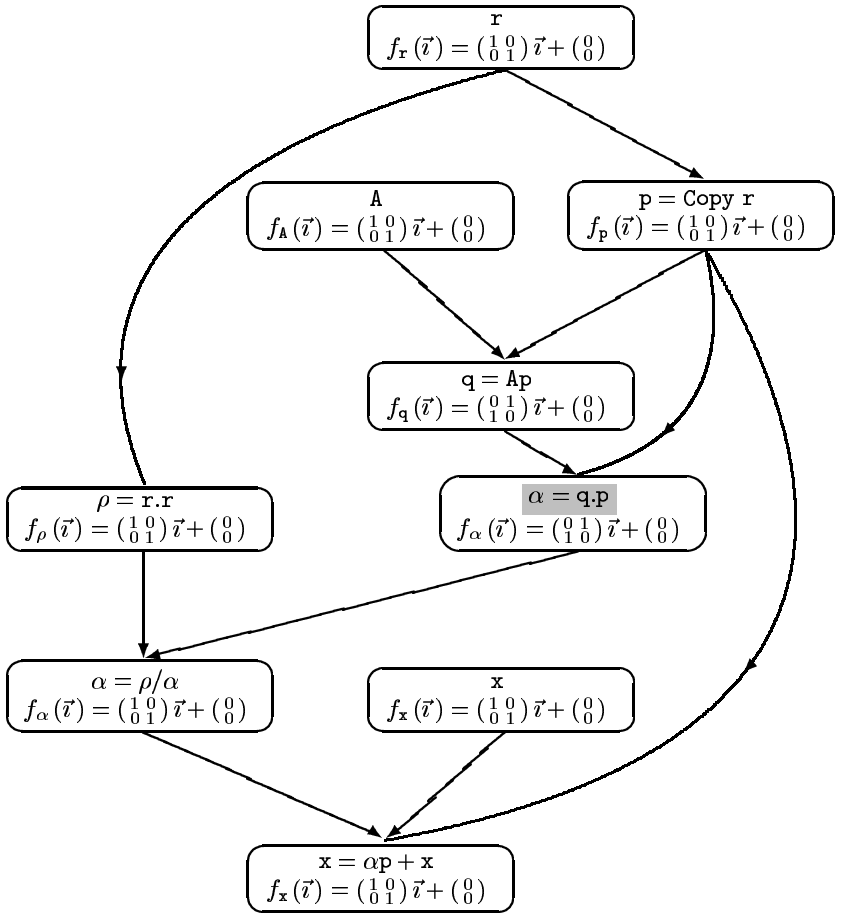
- Execution by-chunks requires regions to grow/shrink through the pipeline.

Optimisations

- Data-driven / demand-driven
- Execution by chunks
  - intermediate data?
- Task farm for stream of frames
- Each filter is actually a specialisation of a generic filter.
  - Specialisation can yield $> \times 10$ speedup.
- Components should be optimised for particular architecture.
  - Up to 30% improvement by recompiling for a particular i686-style architecture.

Case Study 3: Parallel Linear Algebra Solver

- Iterative Solver, implemented by composing parallel basic linear algebra components



Metadata

- Placement constraints
  - enumerate placements?
  - compact representations (affine functions, shown in part in the diagram)
- Performance: Number of communications?

Optimisations

- Data placement optimisation
  - minimise overall redistributions
  - compact representation reduces O-complexity of optimisation

What makes good metadata for optimisation?

- It's all about managing the complexity of optimisation
- Well-designed metadata must
  - factorise a complex optimisation space
  - mean that optimising the composition is cheaper than "opening the boxes" and optimising the whole.
- We must be able to reason about the cost of optimisation.
  - This is made necessary by the staged nature of context
  - Need to decide whether to (re-)optimise at multiple stages.
- Context is staged, so metadata should be "staged"
  - Optimisations must be able to (re-)write metadata
- Domain-specific
  - Using domain-specific semantics *should* facilitate better optimisation

Component Optimisation and Program Generation

- It is unrealistic, and infeasible, to have a component repository that contains optimised implementations for all contexts of component use.
- On-demand program generation is the answer
  - Optimisation(high-level component) $\longrightarrow$ optimised implementation
  - component1 $\circ$ component2 $\longrightarrow$ composite_component
  - this has to include generating metadata
- Require program generation technologies that allow expressing optimisation / code transformation as a separate concern, and which allow formal reasoning
  - AOP?
  - Probably not powerful enough, others?
- This talk has outlined a set of ideas on where the hard work in the optimisation part of WP3 Task 3.3 lies.