ELaSW 2004

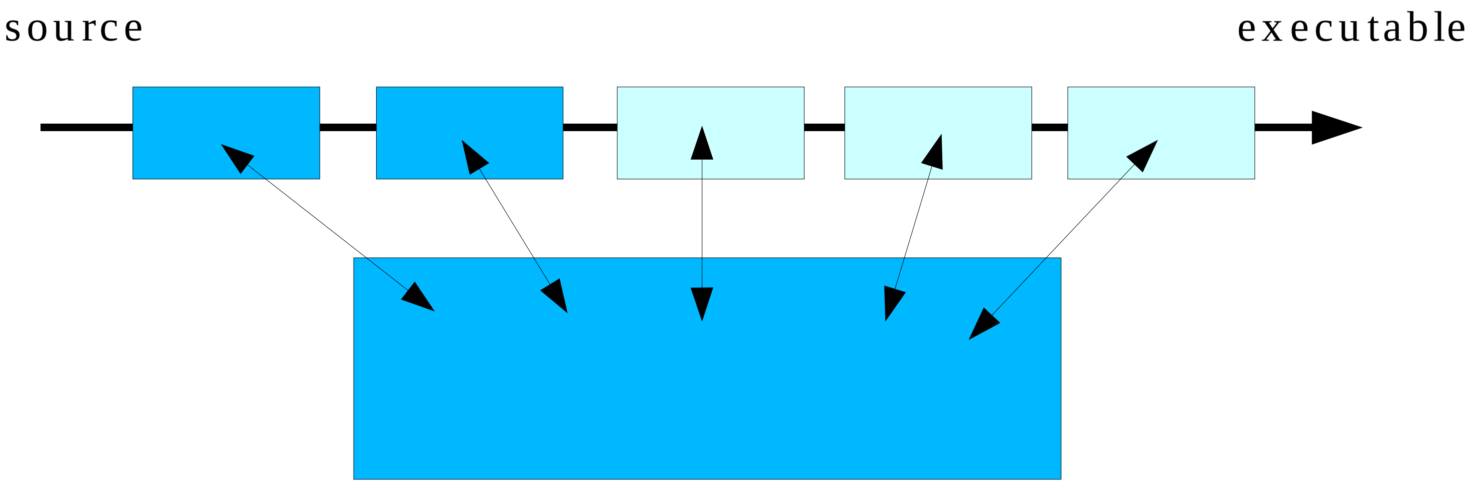
Software Architecture Adaptive Compilers

Jonne Itkonen

ji@mit.jyu.fi

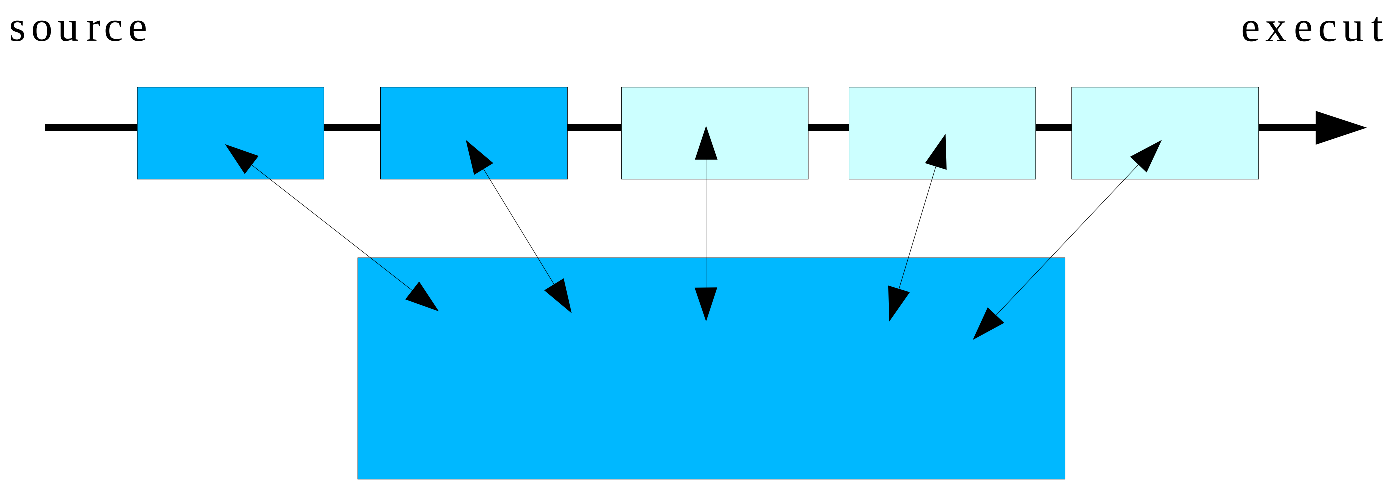
Department of Mathematical Information Technology

Traditional compilers



- hybrid pipe-and-filters and data repository architecture
- command line parameters to change the order or attributes of compilation and optimisation

Compilation sequence

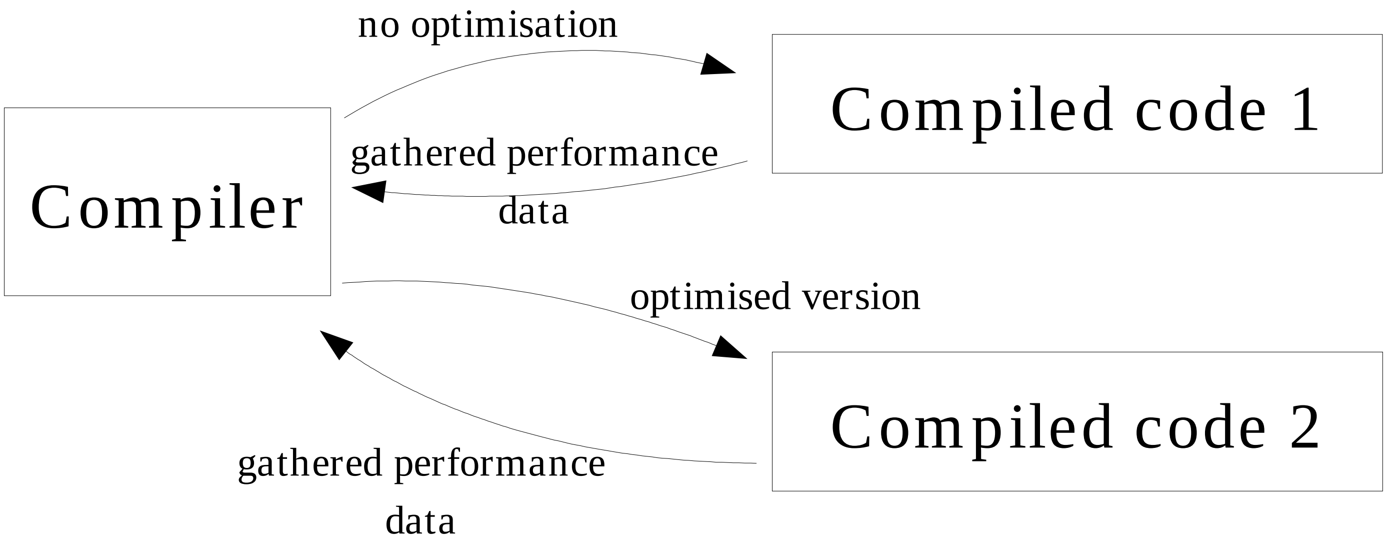


- Ordered list of selected transformations to be applied to the code to be compiled.
  - _ (Cooper et al.)

Adaptive compilers

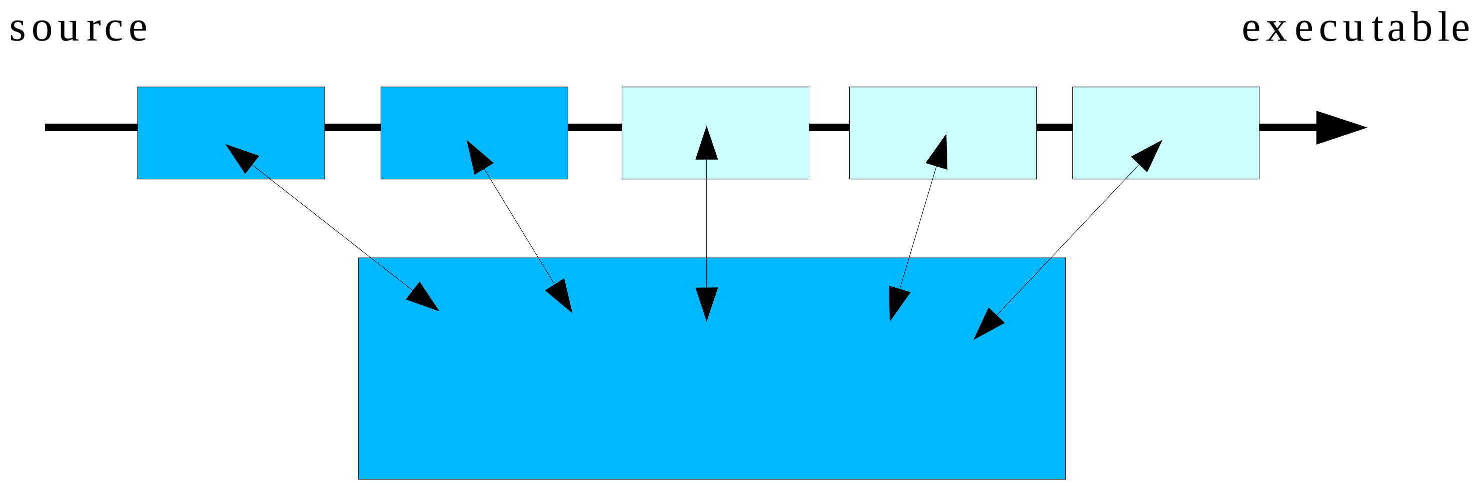
- Change the order of elements in the compilation sequence to get optimum compilation.
- Objectives for example
  - _ smaller size
  - _ faster execution
  - _ smaller power consumption
- Choose from predefined combinations or
- try out different combinations, choose the one with the best results.
- Lazily optimising compilers

Lazy optimisation



1. First time compilation without optimisation.
2. Performance data is gathered and returned to the compiler.
3. Compiler analyses the gathered data and optimises the compilation according this data.
4. Repeat as many time as appropriate.

Software architecture



- "...structures of the system, which comprise software elements, ... properties ..., ... relationships among them." (Bass et al.)
- Compilers have two architectural styles, pipe-and-filters and data repository.

Architecture Description Languages

- Used to
  - describe the architecture and architectural elements
    - components (elements), connectors, ports, roles, attributes, constraints, implementations
  - evaluate and simulate architectures
  - build the software
- XML'tised, but still not enough hype
  - UML?
- My opinion: too little, too hard, too late
  - Why use ADL, when we can write Java? ;-)

ADL example

```
// (Source http://www-2.cs.cmu.edu/~acme/samples/pf-family-def_acme.html)
// Describe a simple pipe-filter family.  This family definition
// demonstrates Acme's ability to specify a family of
// architectures as well as individual architectural instances.
// An Acme family includes a set of component, connector, port and
// role types that define the design vocabulary provided
// by the family.
Family PipeFilterFam = {
  // Declare component types
  // A component type definition in Acme allows you to
  // to define the structure required by the type.  This structure
  // is defined using the same syntax as an instance of a component.
  Component Type FilterT = {
        // All filters define at least two ports
        Ports { stdin; stdout; };
        Property throughput : int;
  };
  // Extend the basic filter type with a subclass (inheritance)
  // Instances of UnixFilterT will have all of the properties and
  // ports of instances of FilterT, plus a stderr port and an
  // implementationFile property
  Component Type UnixFilterT extends FilterT with {
        Port stderr;
        Property implementationFile : String;
  };
  // Declare the pipe connector type.  Like component types,
  // a connector type also describes required structure.
  Connector Type PipeT = {
        Roles { source; sink; };
        Property bufferSize : int;
  };
   // Declare some property types that can be used by systems
   // designed for the PipeFilterFam family
   Property Type StringMsgFormatT = Record [ size:int; msg:String; ];
   Property Type TasksT = enum {sort, transform, split, merge};
};
```

How Lisp can help?

- Have to do some research here!
- ADL in Lisp can be like domain modelling language:
  - use it to describe the system
  - use it to evaluate the system
  - it actually is a Lisp programme, compile it, and you have the executable
- (No examples yet, sorry!)

Compilers written in Lisp

- This idea was discussed a bit on comp.lang.lisp during winter 2003, spring 2004
- Paul F. Dietz' sexpc (http://www.common-lisp.net/project/sexpc/)
- Java compiler? Python? C/C++? Anyone?

```
(defun int main ((int argc) ((* * char) argv))
       ((int i)
        ((* function void (* char)) h)
        ((* char) s "abcdefghi")
        (char c))
       (setq h (ref say_hello_func))
       (h "world")
       (phooey "yo!\n")
       (duff8 (ref c) s 10)
       (cond
         ((not (> argc 1))
          (printf
             "You didn't supply any arguments!\n")
          (printf
             "Usage: %s foo bar baz ...\n"
             (aref argv 0)))
         (else
          (for ((setq i 1)) (< i argc) ((incf i))
               (printf
                  "Your %d%s argument was: %s\n" i
                       (cond ((== i 1) "st")
                             ((== i 2) "nd")
                             ((== i 3) "rd")
                             (else "th"))
                       (aref argv i))))))
       (return 0))
```

Feedback from the executable – Aspect-oriented Programming

- Like in lazily optimising compilers, to get performance data out of the executable.
- ADL/Lisp written aspects wove the data gathering code to the programme.
- AOP and Lisp researched a lot.

What I would like to see...

- Architecture described in Lisp based ADL, rendered same time in some graphical notation (or vice versa), possibly imported from source in some other language
  - McCLIM, Cello
- Code generated in an other language (just for those who don't understand Lisp yet)
  - sexpc
- Code executed, results gathered, compiler adapted, code compiled, ...

Questions?

- Q: Why?
- A1: For the fun!
- A2: Why not? There might be a big innovation lurking behind this... (a feel in the guts)
- A3: Our world is going to be more and more dynamic, why then should we have 'static' compilation?
- Q: Why haven't you done it already?
- A: Just got the idea, but this is not my main interest, at least not yet...

Thank you!