mentor embedded

A MATLAB-to-Target Development Workflow using Sourcery VSIPL++

Stefan Seefeld, Faheem Sheikh, Brooks Moses

September 2012

Comprehensive Solutions for

Android™ ▪ Nucleus® ▪ Linux®

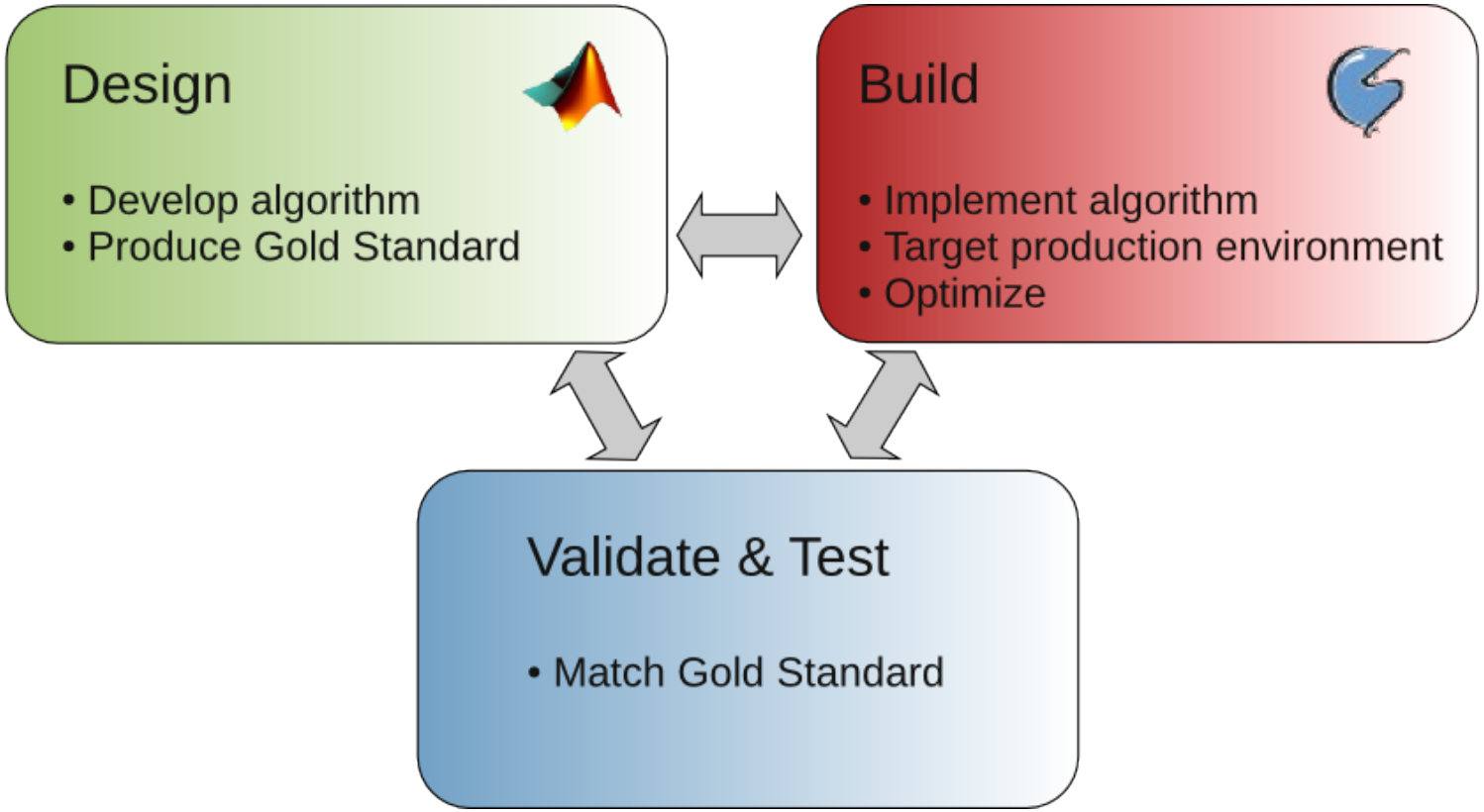
Mobile & Beyond · 2D/3D User Interfaces · Multi-OS · Networking

Mentor Graphics®

Outline

- Introduction - why an integrated workflow ?
- Use cases and details of integration
- Examples and conclusion
- Next steps

Typical Development Workflow



- Prototyping and implementation in two very different environments.
- This leads to redundancy and mis-communication.

Compiled Languages ...

- (...such as C++, C, and Fortran...)
- are fast for computers, but slow for users
- assume that CPU time is more expensive than human time
- don't have interactive capabilities
- have awkward access to plotting, visualization, and system shell

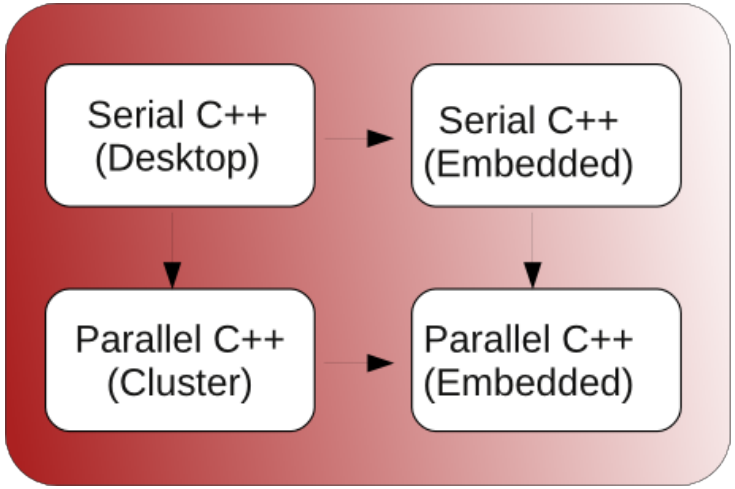
Interactive Computing Environments ...

- (...such as MATLAB, SciPy, and SciLab...)
- are extremely popular with working scientists
  - interactive: matches the exploratory nature of science
  - seamless access to data, algorithms, visualization, etc.
  - great for algorithm development, testing, prototyping, and data analysis
- have poor performance relative to compiled languages

Sourcery VSIPL++ ...

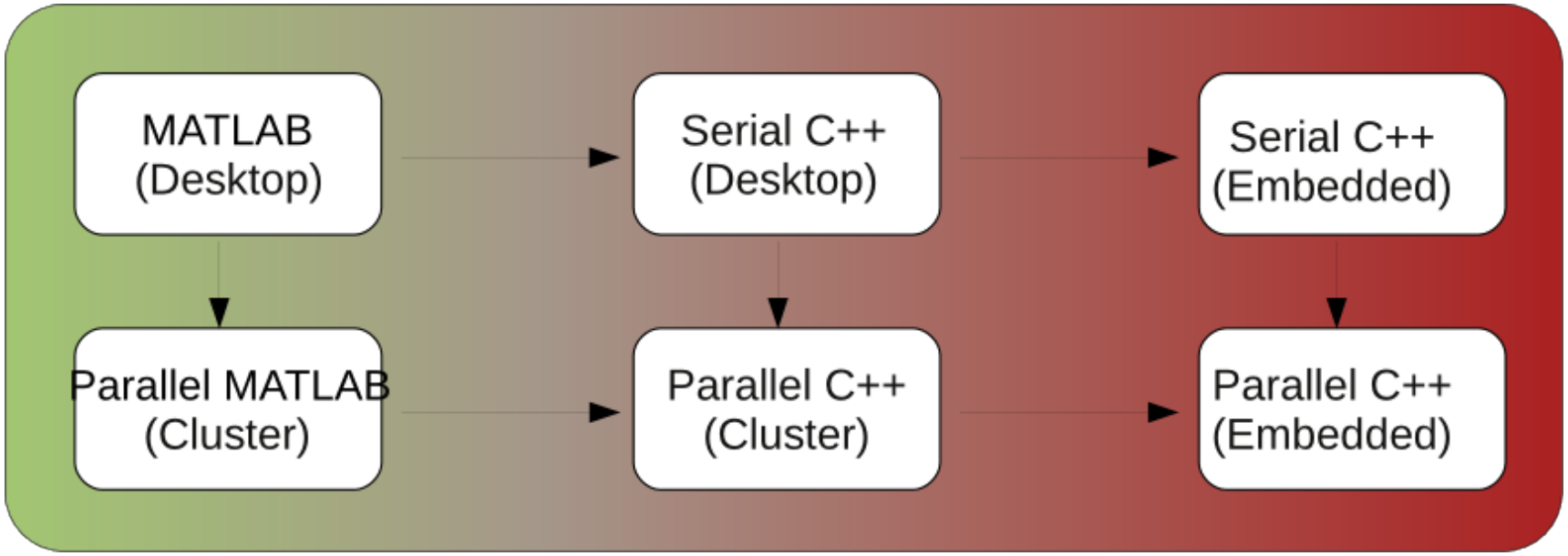
- implements the open VSIPL++ standard
- provides portable performance on a wide range of platforms

  (such as x86, Power, GPGPUs, Cell/B.E....and soon ARM)
- maximizes productivity by virtue of a high-level compact and declarative syntax.
- Develop on desktop, recompile for target platform.



Integrated Development Workflow

- Seamless integration of MATLAB and C++ in hybrid development process
- Maximize code reuse, minimize coding and testing



Multiple Axes of Integration

- sharing the model (API)
- sharing the implementation (backend)
- sharing the process (e.g., testing logic)

Common Data Model and API

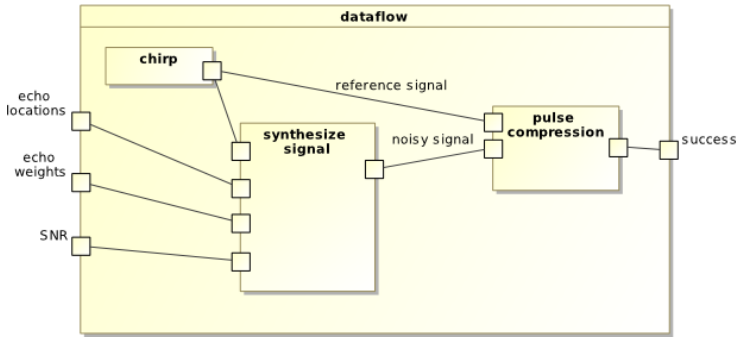
- Using a common Data Model and API removes language barriers
- Allows algorithm developers and software engineers to share a common language / model
- Reduces mapping from MATLAB to C++ to a simple syntactic transcription
- Allows data objects to be transferred between language boundaries

Model Driven Architecture

- The Object Management Group promotes MDA for improved productivity.
- Functionality is defined in an "Platform Independent Model", and later mapped to "Platform-specific Models" using "Language Bindings".
- MATLAB Toolbox API becomes a new Language Binding similar to VSIPL and VSIPL++.
- Focusing on model reduces risk of Gold standard falling out of sync with implementation.

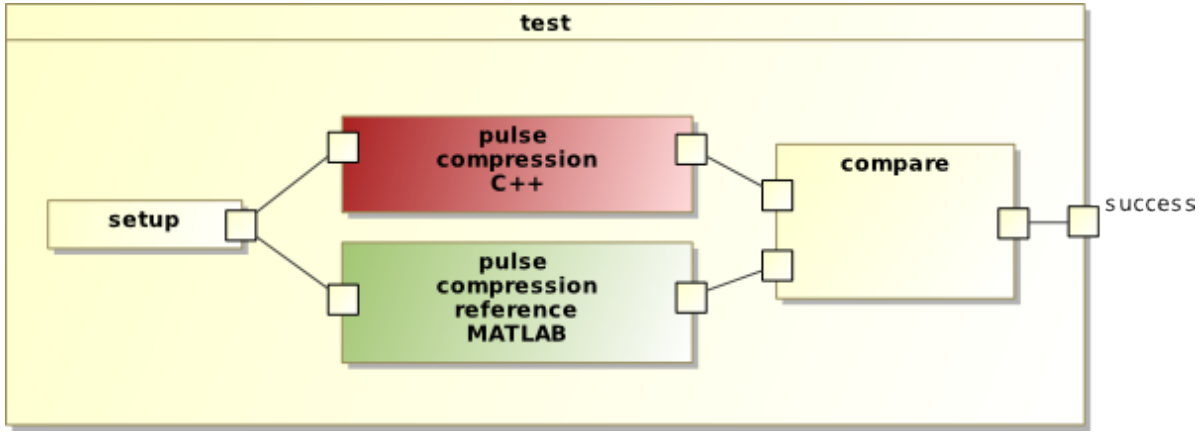
Integrated Testing

- Use Gold standard during testing:
  - No need to rewrite testing logic
  - Help to keep Gold standard up-to-date as requirements and implementation change
- Approach:
  - Develop testing logic in MATLAB
    - Set up environment
    - Run algorithm
    - Validate result



Integrated Testing (cont.)

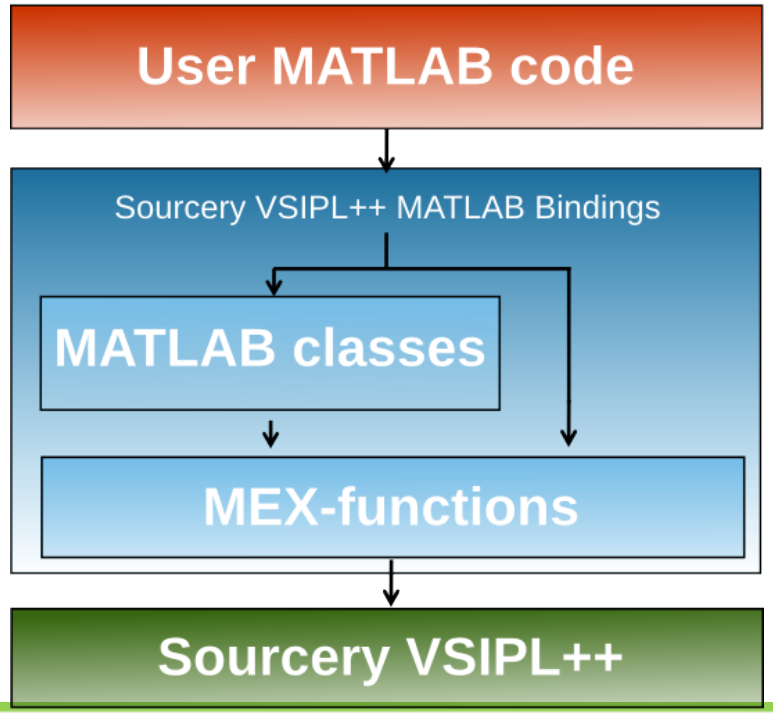
- Approach:
  - Embed MATLAB session into C++ program
  - Share setup logic, compare result of C++ code with Gold standard



- Set up testing environment once, execute in MATLAB and C++, then compare results

Sourcery VSIPL++ MATLAB bindings

- MATLAB Toolkit is implemented with Sourcery VSIPL++
- Views and function objects are implemented with MATLAB classes
- Individual operations call MEX-functions
- MEX-functions call Sourcery VSIPL++ library functions



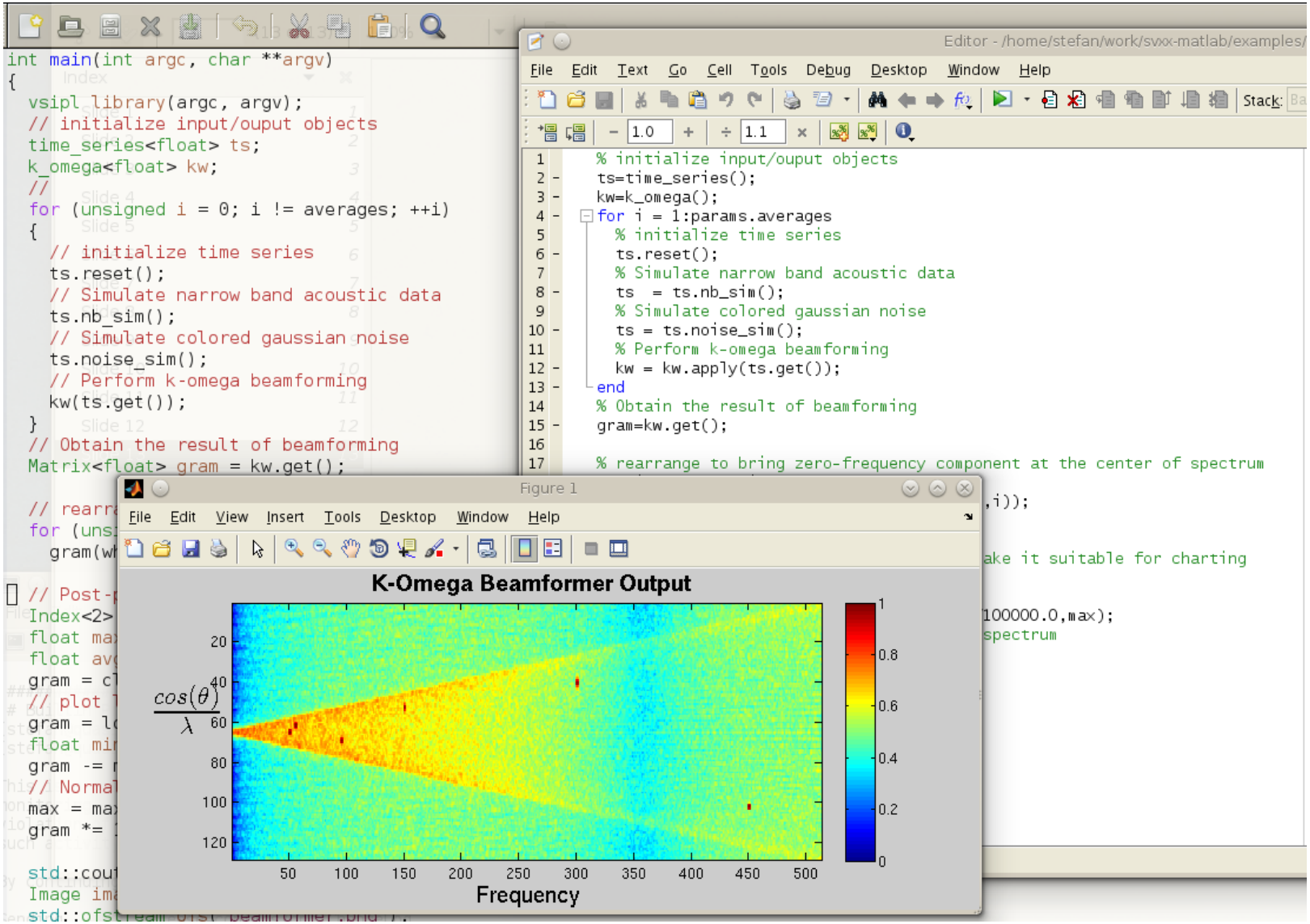
Sourcery VSIPL++ MATLAB bindings (cont.)

- Sharing reduces the differences between environments further
- Allows accelerating MATLAB code using the same accelerator support as C++ code
- Allows to monitor and profile "real" code interactively

Conclusion

- Integrating the prototyping and deployment environments seamlessly has many advantages, resulting in a productivity boost.
- Experiments have shown that algorithms prototyped with MATLAB VSIPL Toolbox can be transcribed into C++ using Sourcery VSIPL++ with minimal effort.
- Example:
  - k-Omega beamformer demo, using ~200 lines of code in MATLAB, transcribed to ~200 lines of C++ code, took only a few hours to implement
  - The only errors made in the process were related to language-specific idiosyncrasies

Conclusion (cont.)



Future work

- Automate the language mapping using source-to-source translation techniques, to map directly from MATLAB VSIPL code to VSIPL++ code.
- Standardize MATLAB VSIPL API, to become a language binding on par with VSIPL and VSIPL++
- Provide more execution feedback to support interactive monitoring and profiling

Questions ?