

# Web Environment for Program Analysis and Transformation onto Reconfigurable Architectures

A.P. Bagly<sup>[0000-0001-9089-4164]</sup>

Southern Federal University, Rostov-on-Don 344006, Russia  
taccessviolation@gmail.com

**Abstract.** Experience of designing different versions of web-based development environment (IDE) for Optimizing parallelizing system and compiler onto reconfigurable architecture is described. Designed system is based on existing tools and frameworks such as Jupyter Notebook and Eclipse Che. Set of requirements for Optimizing parallelizing system components is developed to make it possible to integrate them into web-based development environment accessible through the Internet. Designing portable environment for compiler development, compiler technology demonstration and teaching parallel program development is also described. Newly developed program transformations are shown to be used during program optimizations for FPGA inside the designed web environment. Means of program transformation visualization are described for use with Jupyter Notebook. The work shown demonstrates possibility to organize remote access to library of instruments and tools for program optimizations currently under development that would be convenient for application developers.

**Keywords:** Integrated Development Environment, Program Transformations, Parallelizing Compiler, Containerization, Web IDE, Cloud Computing, FPGA.

## 1 Introduction

During development of a complex software system, like an optimizing compiler as an example, multitude of problems arise that concern overall organization of development process, deployment and usage of developed system. Complexities arise as well with regards to training and inclusion of new developers, organizing outside access to experimental results, like performing demonstrations of particular functions.

Problems that arise are in many ways similar to those that are faced by programmers not familiar with software development for field-programmable gate arrays, as it requires using complex development environments.

At present moment systems that allow to directly transform high level language programs into low-level hardware description for FPGA are either not sufficiently developed or not sufficiently widespread. To alleviate this problem it is sensible to organize remote access to such systems as they are being developed through Internet in form browser accessible integrated development environment, that is complemented with particular extensions that allow developers to iteratively modify their program and resulting low level hardware description, interactively estimate their characteristics. This approach could help solve some of urgent problems that more and more

---

Copyright © 2020 for this paper by its authors.

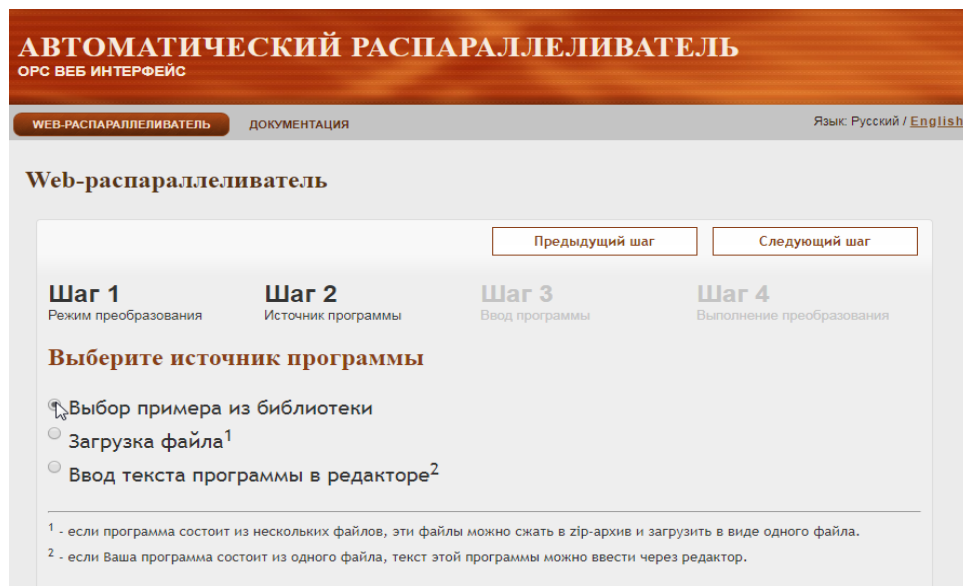
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

application developers are facing in their first attempts to develop software targeting FPGA.

Lets note main problems and goals considered in this work:

1. Rapid deployment of developer environment that includes all necessary tools for new member of developer team or new client for the system.
2. Remote delivery of intuitive and simple dialogue-based interface that allows developer to create hardware description for FPGA from source program.
3. Development of tools to aid demonstration of certain new features of compiler that is being developed.
4. Using proposed web environment and parts of compiler system for teaching software development and parallel programming.
5. Using parts, select functions of system being developed in scientific research which requires ease of access to experimental findings, results and convenience of reproduction.

Earlier some web interfaces were developed for demonstration of Optimizing parallelizing system capabilities [1] and some of its functions [2, 3]. Overall view of first version of such interface is show on Fig. 1.



**Fig. 1.** Window from one of earlier versions of OPS web interface

This kind of interface allows user to upload program source code, choose an action from predefined set of options (program transformations, automatic parallelization, etc.), perform chosen action on remote server and download resulting source code of transformed program.

This way of using web access solves only problem #2, however only partially. User has no ability to directly manage what system does and tweak any parameters, while developer has to create specially crafted scenarios for using the system.

This work states the problem of developing more universal and flexible development environment (software service) that would be accessible remotely through Internet and would have useful properties:

1. Service would be based on currently available tools from cloud computing industry, open source software-as-a-service packages and cloud IDEs.
2. Service would solve all of the stated problems to some extent.
3. Service would not require any extra effort for functionality maintenance from developers as they modify and develop underlying compiler itself while it is being updated.

Compiler onto reconfigurable computer architecture is being developed based on OPS [4] by a team that this work's author is a part of and all stated problems are especially pertinent to that compiler. Therefore approaches that are described here are meant first and foremost to be applied to that project.

Main results of this work include, firstly, new architectural requirements for Optimizing parallelizing system and its components that are gradually adopted in it to allow creation of program optimization tools based on it and creation of education tools for parallel programming and showcasing program optimizations for FPGA programming. Secondly, another outcome of this work is a prototype of web-based development environment for program transformation onto FPGA that is based on OPS and available open-source cloud IDEs.

Further content of this paper is as follows. More detailed requirements for compiler components to facilitate creation of web IDE based on said compiler are described in section "Problem statement". Section "Jupyterlab employment for OPS" describes approach to create interactive development environment for OPS based on interactive browser-based scientific notebooks and C++ language. Section "Results" describes most important modifications that allow to create minimally functional development environment for experimenting with program transformations onto FPGA based on compiler components.

## 2 Problem Statement

In cloud computing industry there exists multitude of widely employed web-oriented development environments with open source code that allow to make extensions for any particular field of study or subject area, for example to support new programming languages, management of supercomputer clusters, etc.

Eclipse Che [5] should be noted in particular, as it has architecture that allows to solve almost all problems that were stated in introduction:

1. Creation and employment of containerised workspaces for system's developers using docker solves rapid deployment problem by easily creating workspaces with repeatable characteristics from recipes.
2. Extensible API for development environment management solves the problem of making convenient demonstration scenarios for particular functions of the system.
3. Possibility to create extensions to all architectural components of the system allows to add new representations and views to demonstrate important functions and visualize results.

However, using this particular software as a basis for specialized cloud development environment requires modifications done to Eclipse Che itself as well as to the compiler.

Generally, modifications necessary would include:

1. Separating compiler pipeline into stages with possibility to export and visualize intermediate results between stages
2. Adding new representations to IDE interface and corresponding methods for workspace API.
3. Creation of set of containers to run compiler and its parts inside predefined environment.
4. Refactor compiler source code to loosen tight relations between modules to make it easier to use them inside cloud-based IDE.

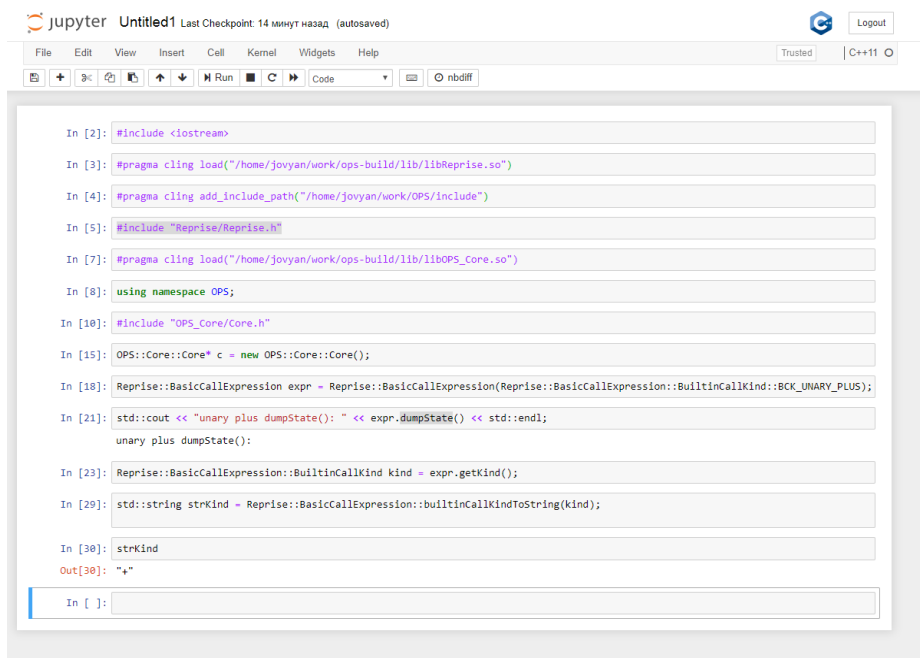
### 3 Jupyterlab Employment for OPS

Jupyterlab [9] is currently the simplest and most well supported tool to organize interactive access to computations with rich data representations and diverse set of compatible technologies. Jupyterlab is a web application and set of language-specific tools that support multitude of programming languages including C++ with use of Xeus-Cling [10]. This allows to offer easy access to all components of OPS using interactive notebooks in web browser, as illustrated on Fig. 2.

Because it is actually possible in Jupyterlab to use interactive notebooks with the same programming language that OPS itself is written in, usage scenarios for various typical use cases and user types could be written easily.

Now to allow different users access to different scenarios it is enough to create set of interactive notebooks that would offer following:

1. Examples of how to use key functions of the system, for example working with internal representation, performing program transformation, which is especially useful for educational purposes.
2. Allow users to perform experiments with examining performance effects and other changes made by different program transformations for sample program using external services for testing, performance analysis, etc.
3. Simple and convenient interface for system's developers to exchange results of their work, data, test results and examples.



```

In [2]: #include <iostream>

In [3]: #pragma clang load("/home/jovyan/work/ops-build/lib/libReprise.so")

In [4]: #pragma clang add_include_path("/home/jovyan/work/OPS/include")

In [5]: #include "Reprise/Reprise.h"

In [7]: #pragma clang load("/home/jovyan/work/ops-build/lib/libOPS_Core.so")

In [8]: using namespace OPS;

In [10]: #include "OPS_Core/Core.h"

In [15]: OPS::Core::Core* c = new OPS::Core::Core();

In [18]: Reprise::BasicCallExpression expr = Reprise::BasicCallExpression(Reprise::BasicCallExpression::BuiltinCallKind::BCK_UNARY_PLUS);

In [21]: std::cout << "unary plus dumpState(): " << expr.dumpState() << std::endl;
unary plus dumpState():

In [23]: Reprise::BasicCallExpression::BuiltinCallKind kind = expr.getKind();

In [29]: std::string strKind = Reprise::BasicCallExpression::builtinCallKindToString(kind);

In [30]: strKind
Out[30]: "+"

In [ ]:

```

Fig. 2. Example of using OPS source code inside Jupyter notebook.

## 4 Visualization

JupyterLab allows to use several cell data formats that allow visualization of data of any type. To visualize results of program transformations it is needed to process following types of data:

1. Program source code in plain text format or with complex notes for operators, variable occurrences, etc., which HTML is the most convenient format to use.
2. Program code with associated graph representation that have variable occurrences, expressions or operators as nodes, which SVG graphics can represent.
3. Tabular data such as test results, performance measurements, source code metrics, all of which could be represented with Markdown formatted text.
4. Graph representations of source code that are particularly complex, such as lattice graph for multidimensional loop nest, that could be represented with a bitmap.

It is most convenient to use HTML to represent source code of initial program and its version after transformation as shown on Fig. 3, which illustrates 2 results of similar source code fragment detection for the purpose of adding new instructions to software CPU. Apart from marking operators by color it is possible to use interactive control elements. Using text data formats such as HTML or SVG makes developing many specific visualizations and passing results around easier.

```

[3]: int MixColumn_AddRoundKey(int *statement,
{
  int word[10][10];
  int ret[32];
  int j;
  register int x;

  for (j = 0; j < nb; j = j + 1)
  {

    ret[(j * 4)] = statement[(j * 4)] << 1;
    if (ret[(j * 4)] >> 8 == 1)
    {
      ret[(j * 4)] = ret[(j * 4)] ^ 283;
    }

    x = statement[(1 + j * 4)];
    x = x ^ x << 1;

    if (x >> 8 == 1)
    {
      ret[(j * 4)] = ret[(j * 4)] ^ (x ^ 283);
    }
    else
    {
      ret[(j * 4)] = ret[(j * 4)] ^ x;
    }
  }

  ret[(j * 4)] = ret[(j * 4)] ^ ((statement[(1 + j * 4)] << 1);
  ret[(1 + j * 4)] = statement[(1 + j * 4)];
  if (ret[(1 + j * 4)] >> 8 == 1)
  {
    ret[(1 + j * 4)] = ret[(1 + j * 4)] ^ 283;
  }

  x = statement[(2 + j * 4)];
  x = x ^ x << 1;

  if (x >> 8 == 1)
  {
    ret[(j * 4)] = ret[(j * 4)] ^ (x ^ 283);
  }
  else
  {
    ret[(j * 4)] = ret[(j * 4)] ^ x;
  }
}

[3]: int MixColumn_AddRoundKey(int *statement, int nb, int n)
{
  int word[10][10];
  int ret[32];
  int j;
  register int x;

  for (j = 0; j < nb; j = j + 1)
  {

    ret[(j * 4)] = statement[(j * 4)] << 1;
    if (ret[(j * 4)] >> 8 == 1)
    {
      ret[(j * 4)] = ret[(j * 4)] ^ 283;
    }
    x = statement[(1 + j * 4)];
    x = x ^ x << 1;
    if (x >> 8 == 1)
    {
      ret[(j * 4)] = ret[(j * 4)] ^ (x ^ 283);
    }
    else
    {
      ret[(j * 4)] = ret[(j * 4)] ^ x;
    }
  }
  ret[(j * 4)] = ret[(j * 4)] ^ ((statement[(2 + j * 4)] << 1);
  ret[(1 + j * 4)] = statement[(1 + j * 4)] << 1;
  if (ret[(1 + j * 4)] >> 8 == 1)
  {
    ret[(1 + j * 4)] = ret[(1 + j * 4)] ^ 283;
  }
  x = statement[(2 + j * 4)];
  x = x ^ x << 1;
  if (x >> 8 == 1)
  {
    ret[(1 + j * 4)] = ret[(1 + j * 4)] ^ (x ^ 283);
  }
  else
  {
    ret[(1 + j * 4)] = ret[(1 + j * 4)] ^ x;
  }
}

```

Fig. 3. Example of using HTML to visualize marked up source code inside JupyterLab

## 5 Program transformation results

During development and testing of optimizing program transformations in Optimizing parallelizing system it is useful to have flexible development environment to test transformations alone and in sequence and estimate their applicability for mapping programs onto FPGA. This environment needs to facilitate display of results, inclusion of samples into documentation. Using JupyterLab with specifically developed extensions that are assembled into integral system satisfied these requirements. Fig. 4 illustrates how program transformations are ran for simple programs, particularly for transformation that forms C function into macro-definition of operation set extension in TCE co-design environment.

```

[26]: "
int main()
{
    int a;
    int b;
    int z;
    a = 0;
    b = 2;
    z = 0;
    z = add(a, b);
    return 0;
}"

[15]: #include <iostream>
std::ostringstream str;
using namespace OPS;
using namespace OPS::Frontend;
using namespace OPS::Reprise;
OPS::Backends::OutToC printer(str)
OPS::Reprise::Declarations& decls = frontend.getProgramUnit().getUnit(0).getGlobals();
OPS::Reprise::DeclIterator<OPS::Reprise::SubroutineDeclaration> it = decls.getFirstSubr();
OPS::Reprise::SubroutineDeclaration& addSub = *it;
OPS::Backends::OutToTCEBehaviour tceBehaviourGenerator(str);
tceBehaviourGenerator.printOperation(addSub, "OPADDINT");

[23]: str.str()

[23]: "#include "OSAL.hh"
OPERATION(OPADDINT)
TRIGGER
int a = INT(1);
int b = INT(2);

    int z;
    z = a + b;
    IO(3) = static_cast<int>(z)

    return true;
END_TRIGGER;
END_OPERATION(OPADDINT)
"

```

**Fig. 4.** Example of running TCE emulator macro-definition generator based on OPS inside Jupyter notebook

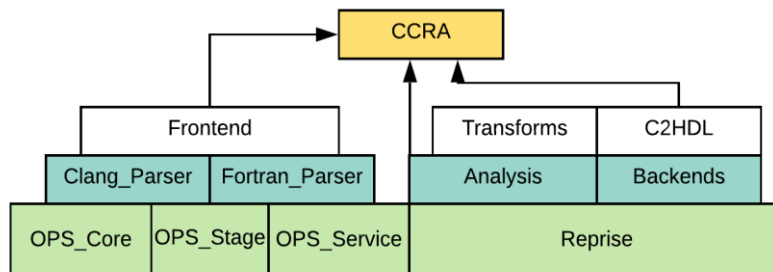
Examples like these allow to expand documentation for OPS compiler, make source code more self-documenting, add to unit tests and integration tests and create easy to use scenarios for different modules, functions and transformation pipelines. This allows to easily perform experiments with parts of the system, develop and test new sequences of program transformations.

## 6 Results

To solve previously stated problems we have introduced following improvements and additions to OPS:

1. Set of container images to facilitate reliable and repeatable builds of OPS itself and of tools based on it, as well as to simplify execution of OPS tools and compiler for sample programs.
2. Extra services to test performance of transformed C language programs similar to earlier developed black box testing system [6], to simulate execution of hardware description for FPGA that was produced by OPS compiler.
3. Compiler internal working process is split into separate stages that could be managed:
  - a. Choosing what parts of program to map onto FPGA could be fully automatic or performed by user via adding pragmas around target fragments.
  - b. Compiler accepts a set of input parameters that describe what actions to perform.
  - c. Possible OPS compiler actions include export of intermediate results for analysis and visualization, generating output program, etc.
4. For several OPS compiler functions corresponding views for data visualization inside IDE are created, for example for examining static profiler results [7] and for common pipeline construction function [8].

Fig. 5 shows a set of main OPS modules that implement independent plugins. This allows to offer access to only limited set of interfaces to the end user, as well as speed up and simplify system build and its deployment.



**Fig. 5.** Set of OPS modular components that include compiler for FPGA (CCRA)

Further work is being done in two main directions:

1. Support for using compiler source code inside cloud-based IDE to make compiler development more convenient.
2. Adding functionality to specialized web IDE that allows to use the compiler and its modules.

Now to start working with OPS source code and OPS compiler it is enough to simply open a given URL inside web browser after which a fresh workspace will be created for the user with full configured development environment. This makes it easy



to introduce users to the system and its source code and lowers amount of time needed for setup, helps in education environment.

## 7 Conclusion

Specialized tools for easy development, employment and educational use of complex software systems could be created on top of modern cloud-based integrated development environments.

To develop functionally sound web-based development environment based on re-targetable compiler it is necessary to solve multiple problems with regards to source code modularity, managing system's external dependencies, making it more portable and developing flexible programming interfaces for accessing its functions. Solving these problems allows to make system development easier as well as allow more convenient use of it, including using it as educational tool for teaching software programming and demonstrating its key capabilities, visualization of experimental data for web IDE users in interactive form.

The reported study was funded by RFBR according to the research project No 18-37-00179.

## References

1. Optimizing parallelizing system URL: [www.ops.rsu.ru](http://www.ops.rsu.ru) (date of access: 25.07.19).
2. Shteinberg, B.Ia., Allazov, A.N., Alymova, E.V., Baglii, A.P., Guda, S.A, Dubrov, D.V., Kravchenko, E.N., Morylev, R.I., Roshal, A.S., Iurushkin, M.V., Shteinberg, R.B.: Web-orientirovannyi avtomaticheskii rasparallelivatel program. In: Parallelnye vychislitelnye tekhnologii (PAVT'2014), Trudy mezhdunarodnoi nauchnoi konferentsii, Rostov-na-Donu (2014).
3. Alymova, E.V., Kravchenko, E.N., Morylev, R.I., Iurushkin, M.V., Shteinberg, B.Ia.: Rasparallelivanie i optimizatsiia programm s pomoshchiu Web-uskoritelia ORS. In: Nauchnyi servis v seti Internet: poisk novykh reshenii, Trudy XIV Mezhdunarodnoi superkompiuternoi konferentsii, Moscow, Izd-vo MGU (2012).
4. Steinberg, B.Y., Bugliy, A.P., Dubrov, D.V., Mikhailuts, Y.V., Steinberg, O.B., Steinberg, R.B.: A Project of Compiler for a Processor with Programmable Accelerator. *Procedia Computer Science*, 101, 435–438 (2016).
5. Localhost is Killing Software Delivery. Codenvy blog <https://blog.codenvy.com/localhost-is-killing-software-delivery-8c93cd49328>, last accessed 2019/11/21.
6. Shteinberg, B.Ia., Alymova, E.V., Baglii, A.P., Morylev, R.I., Nis, Z.Ia., Petrenko, V.V., Shteinberg R.B.: Avtomatizatsiia testirovaniia elementov vysokoproizvoditelnogo programnogo kompleksa. Nauchnyi servis v seti Internet: masshtabiruemost, paralelnost, effektivnost. Trudy Vserossiiskoi superkompiuternoi konferentsii (21-26 sentiabria 2009g., g. Novorossiisk), pp. 287-292, Moscow, Izd-vo MGU (2009).
7. Poluian, S.V.: Profilirovanie i ego primeneniye v dialogovom optimiziruiushchem rasparallelivatele. Nauchnyi servis v seti Internet: superkompiuternye tsentry i zadachi: Trudy Mezhdunarodnoi superkompiuternoi konferentsii, 652-653, Moscow, Izd-vo MGU (2010).
8. Baglii, A.P., Dubrov, D.V., Shteinberg, B.Ia., Shteinberg, R.B.: Povtornoie ispolzovanie resursov pri konveinnykh vychisleniiakh. Nauchnyi servis v seti Internet: trudy XIX Vse-

- rossiiskoi nauchnoi konferentsii, pp. 43-46, Moscow, IPM im. M.V. Keldysha (2017), <https://doi.org/10.20948/abrau-2017>.
9. Kluyver, T. et al.: Jupyter Notebooks – a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, IOS Press Ebooks, pp. 87–90 (2016), <https://doi.org/10.3233/978-1-61499-649-1-87>.
  10. Xeus-cling project on Github, <https://github.com/QuantStack/xeus-cling>, last accessed 2019/11/21.