bbpart user manual

Title	bbpart (CDFG extraction MachSUIF pass)
Author	Nikolaos Kavvadias 2004, 2005, 2006, 2007, 2008, 2009 2010, 2011, 2012, 2013, 2014
Contact	nikos@nkavvadias.com
Website	http://www.nkavvadias.com
Release Date	23 September 2014
Version	1.1.2
Rev. history	
v1.1.2	2014-09-23 Updated header comments in all source files. Added File Listing section in README; added AUTHORS.
v1.1.1	2014-08-12 Added VCG file generation for each procedure (was removed following version **v1.0.1**). Added LICENSE (modified BSD licensing scheme).
v1.1.0	2014-02-24 Changed documentation format to RestructuredText. Added ChangeLog in separate file.
v1.0.9	2006-12-01 Emitter improvements.
v1.0.8	2006-11-29 Option for enabling the marking of "false" dependencies. Not to be used for CFG/SSA SUIFvm.
v1.0.7	2006-09-16 Support for operand data type information.
v1.0.6	2006-09-15 Cumulative update. Several things fixed. ARM target support temporarily removed.
v1.0.1	2004-10-05 Minor additions to README.
v1.0.0	2004-07-17 Initial release.

1. Introduction

`bbpart` is an analysis pass built to be used with the SUIF2/MachSUIF2 compiler infrastructure. This pass generates a graphical representation for the data dependence graphs (which come in the form of DAGs) of the basic blocks found in a given ANSI C source file.

The DAG for each basic block is depicted in the VCG (Visualization and Compiler Graph) format. A link to VCG can be found here: http://rw4.cs.uni-sb.de/~sander/html/gsvcg1.html

A naming convention has been adopted for each generated file as seen below:

```
<procedure name>_<basic block number>.vcg
```

where `<procedure name>` is a C procedure, `<basic block number>` enumerates the basic blocks in the procedure starting from zero, and vcg is the default extension for VCG files.

This pass works for the SUIF virtual machine instruction set (SUIFvm) and for its most common 2 or 3 addressing modes. A more general version of `bbpart` applicable to all SUIFvm addressing modes will eventually be released in the not so distant future. Currently, a new internal version of `bbpart` is being developed which is more elegant than the current `bbpart` since it uses the `map_opnds API` function of MachSUIF. The problem is that it is not very stable, and programs hang right after results have been produced for only a few basic blocks in the program. I believe this is due to a de-efficiency of the MachSUIF API.

If nothing is changed in the default `do_lower` pass of Machine SUIF (dismantling to simpler objects of several IR objects present in the corresponding SUIF2 representation), `bbpart` should not experience any problems with portable C application codes.

The `bbpart` pass has been tested with MachSUIF 2.02.07.15.

2. File listing

The `bbpart` distribution includes the following files:

/bbpart	Top-level directory
AUTHORS	List of `bbpart` authors.
LICENSE	The modified BSD license governs `bbpart`.
README.rst	This file.
README.html	HTML version of README.
README.pdf	PDF version of README.
VERSION	Current version of the project sources.
bbpart.cpp	Implementation of the `bbpart` analysis pass.
bbpart.h	C++ header file containing declarations and prototypes for the above.
rst2docs.sh	Bash script for generating the HTML and PDF versions of the documentation (README).
suif_main.cpp	Entry point for building the standalone program `do_bbpart` that implements the pass.

suif_pass.cpp	Define the SUIF pass built as the dynamically loadable library `libbbpart.so`.
suif_main.h	C++ header file for the above.
utils.h	C header file with implementations of auxiliary functions.

3. Installation

Unpack the `bbpart` archive wherever you like, e.g. in `$MACHSUIFHOME/cfg/bbpart`. You don't need to modify anything in the Makefile, if you have a working MachSUIF 2 installation.

The program binary (`do_bbpart`) will be installed at `$NCIHOME/bin` and the shared library (`libbbpart.so`) at `$NCIHOME/solib`, where `NCIHOME` is the SUIF 2 top-level directory.

4. Usage details

The pass accepts an input file in CFG form to operate. You don't have to define any output files since their naming convention has been hardwired.

As said above, the bbpart pass is applied on the CFG representation of the input program. Which means that you have to run a sequence of transformation passes on the C program. These correspond to transformation and optimization phases of the modular SUIF/MachSUIF compiler. You can prepare something like the following script (it is just a minimalistic script to get you working), that can be run e.g. from `csh`:

```
c2s $1.c $1.suif
do_lower $1.suif $1.lsf
do_s2m $1.lsf $1.svm
do_il2cfg -break_at_call $1.svm $1.afg
do_bbpart $1.afg
echo "Done with $1"
```

Then if this is called `run_bbpart` you can run it on `filename.c` as follows:

```
$ ./run_bbpart filename
```

For the C program `test.c` containing a main procedure with 5 basic blocks and an abs procedure with 3 basic blocks the following files will be generated:

```
abs_0.vcg
abs_1.vcg
abs_2.vcg
main_0.vcg
main_1.vcg
main_2.vcg
main_3.vcg
main_4.vcg
```

Usage synopsys:

" do_bbpart [options] test.afg"

where options can be one or more of the following:

-dt enable the production of operand data type information. The following data types are supported: {v0, u8, u16, u32, u64, s8, s16, s32, s64, f32, f64, f128, p32, p64}.

-mark_false_deps enable the marking of "false" operand dependencies. This option should not be used for CFG/SSA SUIFvm.

-global_symbol_table generation of global symbol table entries.

5. Notes

If you use `bbpart` in any publication, please give a reference to the following paper:

Nikolaos Kavvadias and Spiridon Nikolaidis, "Application Analysis with Integrated Identification of Complex Instructions for Configurable Processors," Proc. of the 14th Intl. Workshop on Power and Timing Modeling, Optimization and Simulation, pp. 633-642, September 15-17, 2004, Santorini, Greece.

This paper discusses a prototype application analysis flow with MachSUIF where `bbpart` is used as a CDFG extractor.