

WHIRL SSA Enhancements

Jian-Xin Lai

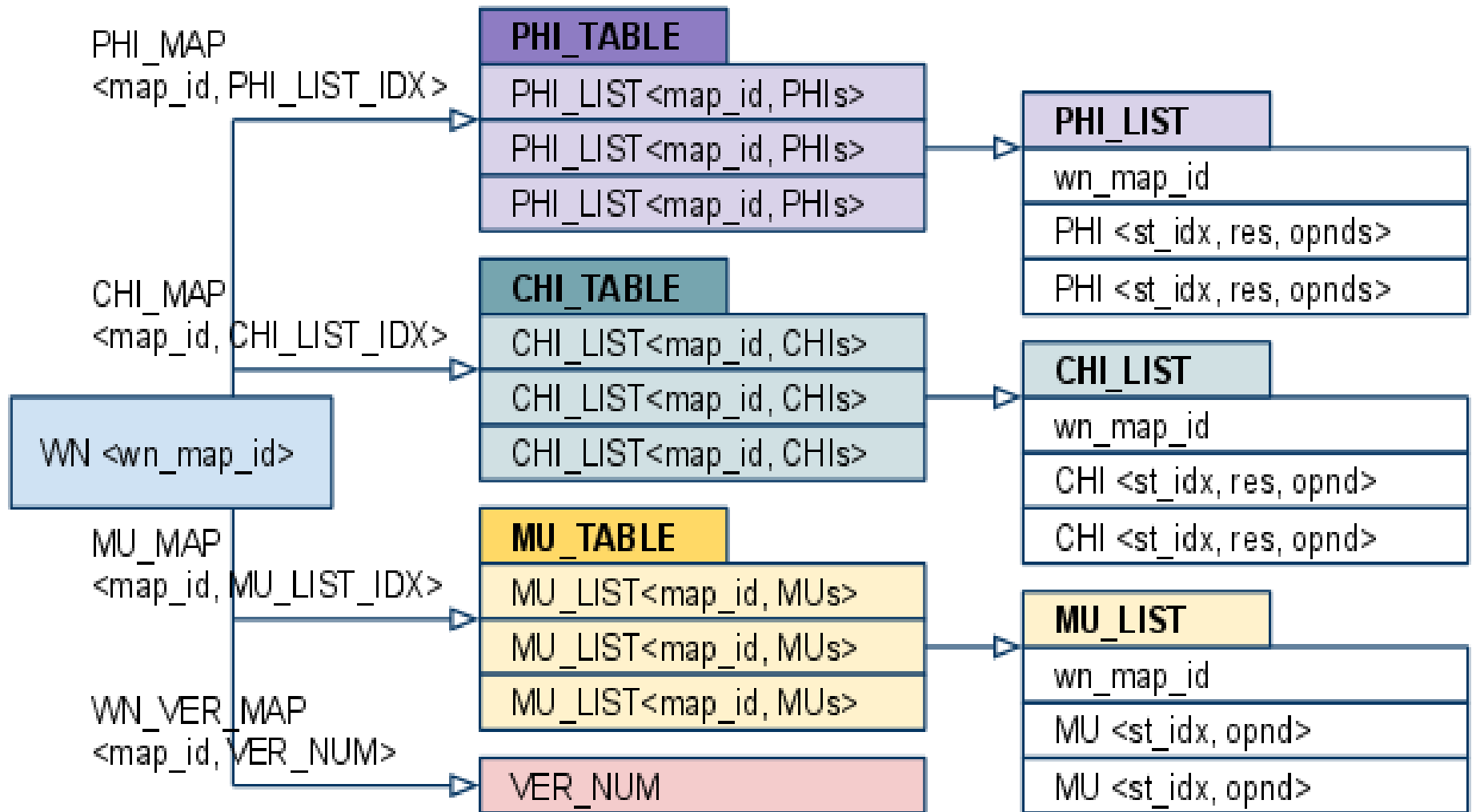
Agenda

- **Introduction to WHIRL SSA**
- WHIRL CFG
- UD/DU-chain
- WSSA updater
- WSSA standalone builder

WSSA: Design Decisions

- Side-Data Structure
 - No changes to the existing components
 - No new operators introduced to WHIRL
- Memory SSA
 - Aliases are represented by χ and μ
 - Virtual symbols for indirect loads/stores
- Available from H WHIRL to L WHIRL
 - M/L WHIRL SSA is well tested
- Live-ranges of the versions (of the same variable) don't overlap

WSSA Tables and Maps



WSSA: An example

```
int *a;
Int sum;
int foo(int n) {
    int i;
    for(i=0; i<n; i++)
        sum += a[i];
    return sum;
}
```

WSSA SYMTAB TABLE

n	ST	ST_IDX
sum	ST	ST_IDX
a	ST	ST_IDX
i	ST	ST_IDX
preg	ST	ST_IDX
_v_def	VSYM	default
_v_ret	VSYM	return

χ TABLE

nv2=χ(nv0)
sumv2=χ(sumv0)
av2=χ(av0)
<u>_v_def_v2=χ(_v_def_v0)</u>
<u>_v_ret_v2=χ(_v_ret_v0)</u>
<u>_v_def_v5=χ(_v_def_v3)</u>
<u>_v_ret_v5=χ(_v_ret_v3)</u>

φ TABLE

sumv3=φ(sumv2, sumv5)
iv2=φ(iv1, iv3)
<u>_v_def_v3=φ(...v2, ...v5)</u>
<u>_v_ret_v3=φ(...v2, ...v5)</u>

μ TABLE

<u>μ(_v_def_v3)</u>
<u>μ(_v_ret_v3)</u>
<u>μ(sumv3)</u>

```
FUNC_ENTRY <1,50,foo>
  IDNAME 0 <2,1,n>
  BODY
    BLOCK
      DO_LOOP
        IDNAME 49 <1,i>
        INIT
          U4INTCONST 0
          I4STID 49 <1,4,i> T<4,predef_I4,4>
        COMP
          I4I4LDID 0 <2,1,n> T<4,predef_I4,4>
          I4I4LDID 49 <1,4,i> T<4,predef_I4,4>
          I4I4GT
          INCR
            I4I4LDID 49 <1,4,i> T<4,predef_I4,4>
            U4INTCONST 1
            I4ADD
              I4STID 49 <1,4,i> T<4,predef_I4,4>
          BODY
            BLOCK
              U4U4LDID 0 <1,51,a> T<53,anon_ptr..4>
              I4I4LDID 49 <1,4,i> T<4,predef_I4,4>
              U4INTCONST 4
              U4MPY
              U4ADD
                I4I4ILOAD 0 T<4,predef_I4,4> T<53,anon_ptr..4>
                I4I4LDID 0 <2,3,sum> T<4,predef_I4,4>
              I4ADD
                I4STID 0 <2,3,sum> T<4,predef_I4,4>
            END_BLOCK
          I4I4LDID 0 <2,3,sum> T<4,predef_I4,4>
          I4STID 1 <1,4,reg_I4v3> T<4,predef_I4,4>
        RETURN
      END_BLOCK
```

VER MAP

iv1
nv2
iv2
iv2
iv3
av2
iv2
sumv3
sumv5
sumv3
pregv3

M/L WHIRL SSA

- Generated by MainOPT emitter

Data Structure	HSSA	WHIRL SSA
Symbol entry	AUX_STAB_ENTRY	WSSA::WST_Symbol_Entry
Version entry	VER_STAB_ENTRY	WSSA::WST_Version_Entry
ϕ node	PHI_NODE	WSSA::PHI_NODE
χ node	CHI_NODE	WSSA::CHI_NODE
μ node	MU_NODE	WSSA::MU_NODE
IR	CODEREP, STMTREP	WN*

- Run an extra renaming phase to correct the version numbers
- ϕ nodes can only be found on OPR_LABEL

Agenda

- Introduction to WHIRL SSA
- **WHIRL CFG**
- UD/DU-chain
- WSSA updater
- WSSA standalone builder

WHIRL CFG: Design Decisions

- Side-Data Structure
 - Based on WHIRL but no changes to WHIRL
 - No changes to existing components
- Available from H WHIRL to L WHIRL
 - M/L WHIRL SSA is well tested
- Instance managed by O64_Driver
- Multiple purposes
 - CFG functionality
 - Provide control flow information
 - Provide dom/post-dom/dom-frontier/control-dep information
 - Auxiliary WHIRL functionality
 - Built-in parent-child map
 - Utilities to get the parent WHIRL BLOCK, REGION, SCF, etc

WN_CFG class

- Represent control flow graph
 - Build and verify CFG
 - Repeat the same process of building CFG to verify it
 - Manage Basic Blocks
 - Create, delete, merge, split, connect, disconnect BBs
 - Bi-directional mapping between WN* to BB*
 - Map label number to BB*
 - Dummy entry and exit
 - Dummy entry connects to all real entries (FUNC_ENTRY, ALTENTRY, EH handler)
 - All real exits (RETURN, RETURN_VAL statements) reach dummy exit

WN_CFG::BB_NODE

- Represent a Basic Block in CFG
- Manage the statements in BB
 - First_stmt, Last_stmt
 - stmt_iterator
- Predecessor and successor BB iterators
- IDom and iterator to BBs immediately dominated
- IPDom and iterator to BBs immediately post-dominated
- DF and CD iterators

WCFG Iterators

- `WN_CFG::bb_iterators`
 - Traverse all BBs in their creation order
- `DFS_ITERATOR`
 - Traverse all BBs in depth-first order
- `DOM_WALKER_HELPER`
 - Take a functor as parameter and traverse the DOM tree in depth-first order

Agenda

- Introduction to WHIRL SSA
- WHIRL CFG
- **UD/DU-chain**
- WSSA updater
- WSSA standalone builder

UD/DU-chain

- A version can be ...
 - defined by WN, ϕ , or χ
 - used in WN, ϕ , χ or μ
- UD-chain embedded in version entries
 - WST_Version_Entry contains
 - WN* where it is defined
 - how it is defined
- DU-chain is built *on demand*

DU Manager and Iterator

- **WSSA_DU_MANAGER**
 - DU-chain is not updatable
 - created on demand
 - discarded after transformation
- **WSSA_DU_ITERATOR**
 - Take a WN^* as start point
 - The WN^* may define multiple versions for different symbols
 - Defined by WN itself, χ , or ϕ if the WN is `OPR_LABEL`
 - The iterator will traverse all uses of these versions

Agenda

- Introduction to WHIRL SSA
- WHIRL CFG
- UD/DU-chain
- **WSSA updater**
- WSSA standalone builder

WSSA_UPDATER

- Updates SSA incrementally
 - Transformation is simple
- Compile-time
 - Much faster than rebuilding the whole SSA
- Changing CFG is not supported
- Three cases
 - Single statement
 - Single BB
 - Multiple BBs

WSSA_UPDATER: Cases 1

- WSSA_UPDATER::Enter_stmt(WN* stmt);
- Single statement
- Requirements
 - The statement is inserted into the WHIRL tree
 - If the statement uses an existing version
 - The version should have been attached to the statement
 - If the statement defines a new version
 - The new version is never used by existing statements
- Actions
 - If statement defines new version (STID),
 - Create a WST_Symbol_Entry unless one exists already
 - Create a WST_Version_Entry

WSSA_UPDATER: Cases 2

- WSSA_UPDATER::Rename_single_BB(BB_NODE* node);
- Single BB
- Requirements
 - The WHIRL tree is already updated
 - If the statement has upwards an exposed use
 - The version must have been attached to the statement
 - If new versions are defined
 - The new versions are never used outside the current BB
- Actions
 - Initiate the rename stack
 - Rename all versions in the BB

WSSA_UPDATER: Cases 3

- `WSSA_UPDATER::Rename_BB(BB_NODE* node);`
- Multiple BBs
- Requirements
 - The WHIRL tree is already updated
 - If the statement has an upward exposed use
 - The version is attached to the statement
 - If a new version is defined
 - The new versions can be used outside the current BB
- Actions
 - Initiate the rename stack
 - rename all versions in the BBs dominated by the given BB

Agenda

- Introduction to WHIRL SSA
- WHIRL CFG
- UD/DU-chain
- WSSA updater
- **WSSA standalone builder**

WSSA_BUILDER: Concept

- Builds SSA from scratch using ALIAS_MANAGER
 - Invoked when WSSA is hard to update
- Memory SSA, Minimal SSA, non-overlapping live ranges
- Designed to work with the new alias analysis
 - AliasTag
 - An ID associated with a WN accessing memory and variables
 - An ID contained in POINTS_TO, used to query the AliasAnalyzer results
 - Alias_Analyzer
 - A new class that provides the interface to the alias analysis results

WSSA_BUILDER: Steps

- 3 steps
 - Mu/Chi/virtual symbol generation
 - ϕ insertion
 - Renaming
- Step 2 and 3 are the same as in the Cytron's SSA paper
 - Minimal SSA
- The WSSA builder and verifier share most of the code

WSSA_BUILDER: Memory SSA

- Map AliasTag to virtual symbol in WSSA Symbol Table (WST)
 - 1:1 mapping (unique vsym)
 - Best accuracy and worst scalability
 - N:1 mapping (default vsym)
 - Best scalability and worst accuracy
 - WHIRL_SSA_MANAGER maintains the map
- Generate Mu/Chi list by AliasTagList
 - Query ALIAS_MANAGER for AliasTagList
 - Query WHIRL_SSA_MANAGER to map AliasTag to WST
 - Generate Mu/Chi nodes for the WST entries

WSSA_BUILDER: Status

- Not finished yet
- The interfaces in the slides may change
- Target date: The end of May. 2011.

Thanks 😊
Q&A?

Backup Slides