

# Tutorial on NVIDIA's Open64 Sources

by Mike Murphy

11/06

# Outline

- **What it is**
- Where it is
- How to build it
- How to use it
- How to debug it
- How to change it
- Future work

# What it is

- nvopencc is a variant of the open-source Open64 compiler that targets NVIDIA's virtual assembly PTX.
- nvopencc is invoked by nvcc, which does a preprocessing pass with cudafe, then calls nvopencc to produce PTX, which is then fed into OCG to produce SASS.

# What it is - Definitions

- **Open64:** [www.open64.net](http://www.open64.net)  
sources: `sw/compiler/gpgpu/open64/src`  
docs: `<src>/doc/howto-debug-compiler`  
`www.open64.net/documentation/manuals.html`
- **nvCC:** `sw/compiler/gpgpu/doc/nvcc.doc`
- **PTX:** `sw/compiler/gpgpu/doc/spec/ptx_isa_beta.doc`

# Subset of Open64

- supports C, not C++ or FORTRAN
- no Inter-Procedural Analysis
- no Loop Nest Optimization
- no preprocessing or linking

# 3 sub-executables

- Front end (gfec)
  - based on gcc, produces WHIRL IR
- Inliner (inline)
  - inlines all calls
- Back end (be)
  - optimizes and lowers WHIRL into PTX

# Back End phases

- VHO (Very High Optimizer)
  - switch -> if/else
  - struct copies -> field copies
- WOPT (Whirl OPTimizer)
- CG (Code Generator)

# WOPT

- translates WHIRL into SSA (Static Single Assignment) form then back to WHIRL
- PreOpt => MainOpt => RVI
- be/opt/opt\_main.cxx lists main papers for algorithms
  - constant folding
  - copy propagation
  - dead code elimination
  - full and partial redundancy elimination
  - control flow optimization
  - register variable identification
  - strength reduction
  - induction variable recognition and elimination
  - code motion
  - alias analysis

# CG

- expand WHIRL into PTX
- assign virtual registers
- convert 32-bit ops into 16-bit ops
- rematerialize GRF loads to reduce live-ranges
- combine contiguous load/stores into vectors
- emit PTX
- no scheduling
- no “real” register allocation
- relies on OCG

# Changes from default Open64

- ported to new target PTX
- host work to build on windows
- new intrinsics
- memory spaces
- optimizing struct copies
- tuning WOPT optimizations
- CG optimizations: vectors, rematerializing, 16-bit conversion

# Outline

- **What it is**
- **Where it is**
- How to build it
- How to use it
- How to debug it
- How to change it
- Future work

# Source Directories

- target-specific subdirectories like NVISA or x8664
- `ifdef TARG_NVISA // NVISA == PTX`
- `sw/compiler/gpgpu/open64/src/*`
  - `be/be` - backend driver
  - `be/cg` - code generator
  - `be/com` - common/shared files
  - `be/lno` - loop nest optimizer
  - `be/opt` - whirl optimizer
  - `be/region`- region utilities
  - `be/vho` - very high whirl optimizer
  - `common/com` - main common files (WHIRL/symtab)
  - `common/targ_info` - target description
  - `common/util` - utilities

# more source directories

- doc - howto-debug document
- driver - nvopenc driver
- gccfe - C front end takes gnu IR->WHIRL
- gccfe/gnu - actual gcc code
- include - headers used by open64
- ipa - inter-procedural analysis
- ir\_tools - ir\_b2a for dumping whirl files
- libdwarf - dwarf library
- libdwarf/dwarfdump – utility to dump dwarf info
- libelf - elf library
- libelfutil - extra elf utilities
- libiberty - gnu utilities
- linux/make - gcommon{defs,rules} included by all makefiles

# build directories

- `targia32*` - where build compiler on ia32 host
  - `targia32_nvisa` - nvisa target on linux
  - `targia32_x8664` - x86 target on linux
  - `targia32gw_nvisa` - nvisa target on mingw
  - `targia32cyg_nvisa` - nvisa target on cygwin
  - `*_rel` directories for non-debug release builds
- 
- installs in `export/*/open64/bin/nvopencc`
  - `nvopencc` looks in `../lib` for `gfec/inline/be`
  - `export/*/bin/nvcc.profile` has path to `nvopencc`

# Outline

- What it is
- Where it is
- **How to build it**
- How to use it
- How to debug it
- How to change it
- Future work

# building on linux

- `cd sw/compiler/gpgpu; make open64_install`
- `cd open64/src/targia32_nvisa; make`
- `cd targia32_nvisa/libcgc; make expand.o`
- build directories != source directories
- `<src>/Makefile.gbase` for each build dir

# building on windows

- same as linux, but need recent cygwin
  - `sw/tools/win32/cygnus/2006`
- uses mingw so resulting executables can run on systems that don't have cygwin
- backend uses static libraries rather than dlls/dsos

# Outline

- What it is
- Where it is
- How to build it
- **How to use it**
- How to debug it
- How to change it
- Future work

# from nvcc

- `nvcc -keep <file>.cu`
  - produces `<file>.cpp3.i`
  - input to `nvopencc`, creating `<file>.ptx`
- `--opencc-options <option>`
  - passes `<option>` to `nvopencc`
  - e.g. `--opencc-options -Wfib\\,-ttmsc:0x40`
- `setenv OPENCC_FLAGS <option>`

# nvopencc directly

`nvopencc -show -keep x.i`

`<path>/lib/gfec -O2 -quiet -m32 -fpreprocessed -fbuiltin x.i -o x.B`

`<path>/lib/inline -O2 -INLINE:all -TARG:abi=n32 -fB,x.B -fI,x.I x.i`

`<path>/lib/be -PHASE:w:c -O2 -TARG:abi=n32 -LANG:=ansi_c -fB,x.I`

`-s -fs,x.ptx x.i`

x.B and x.I (or x.BI) are elf files containing WHIRL

`-W{fib},<option>`

`-Wb,<option>` passes option to back end

Group option syntax: `-WOPT:<flag>=<val>:<flag2>`

# Outline

- What it is
- Where it is
- How to build it
- How to use it
- **How to debug it**
- How to change it
- Future work

# ir\_b2a

- `targ*/ir_tools/ir_b2a` (Binary2Ascii)
- `ir_b2a x.B` will dump the WHIRL
- `ir_b2a -st x.B` will dump WHIRL and symbol table

# ir\_b2a example

```
int increment (int i)
{
    return ++i;
}
ir_b2a produces:
LOC 0 0 source files: 1    "c:\test/incr.i"
LOC 1 1 int increment (int i)
LOC 1 2 {
FUNC_ENTRY <1,32,increment>
IDNAME 0 <2,1,%parm_i>
BODY
BLOCK
END_BLOCK
BLOCK
END_BLOCK
BLOCK
PRAGMA 0 120 <null-st> 0 (0x0) # PREAMBLE_END
LOC 1 3    return ++i;
BLOCK
I4I4LDID 0 <2,1,%parm_i> T<4,.,predef_I4,4>
I4INTCONST 1 (0x1)
I4ADD
I4STID 0 <2,1,%parm_i> T<4,.,predef_I4,4>
END_BLOCK
I4I4LDID 0 <2,1,%parm_i> T<4,.,predef_I4,4>
I4COMMA
I4RETURN_VAL
END_BLOCK
```

# ir\_b2a example explained

- LOC refers to source position (LOCation).
- The FUNC\_ENTRY has one parameter: IDNAME 0 <2,1,%parm\_i>
- The later LDID is a load of this parameter.
- The <> gives a reference to the symbol table (level 2, index 1, name %parm\_i).
- The symbol table usually has two levels: globals at level 1, and locals at level 2.
- There is a separate global table of types, which are the T<4,.predef\_I4,4> references, which means type #4, named predef\_I4, alignment 4.
- The I4 in the type and opcodes is a predefined "mtype": signed 4-byte integer.
  - Open64 types are in terms of bytes, whereas in PTX they are in bits,
- The I4I4LDID 0 <symbol> <type> says to load an I4 from offset 0 of <symbol>.
- The first couple of empty BLOCKs are for pragmas; the third BLOCK has the list of statements, which in this case is just a store (STID).
- The code is printed in postfix order, so the child of STID is ADD, which has two kids, a LDID of parm\_i and the constant 1.

# traces

- traces from `-t*` options are put in `.t` files
- see `src/doc/howto-debug-compiler`
- `-tr<phase>` gives IR dump after phase
- `-ts<phase>` gives symbol table after phase
- `-tt<phase>:<val>` gives trace within phase
- `-Wb,-trvho,-trlow`
- `-Wb,-ttop:0xffffffff,`
- `-Wb,-ttexp:7,-trlra,-trebo`

# adding a trace

```
if (Get_Trace(TP_CGEXP, 0x800)) {  
    fprintf (TFile, "new trace\n");  
}
```

```
-Wb,-texp:0x800
```

# adding a flag

- for `–WOPT:<flag>` add to `common/com/config_wopt.cxx`

```
{ OVK_BOOL, OV_VISIBLE, TRUE, "estr_outer_loop", "",  
  0, 0, 0, &WOPT_Enable_Estr_Outer_Loop, NULL },
```

```
if (WOPT_Enable_Estr_Outer_Loop)
```

- for `–CG:<flag>` add to `be/cg/cgdriver.cxx`

# DevWarns and Assertions

- `DevWarn("why am I here?");`
- `-Wfib,-ttmsc:0x40` to turn on DevWarns
- `FmtAssert(condition, ("message"));`

# debugging

- builds with gcc, so use gdb
- can set breakpoint in `Fail_FmtAssertion` or `DevWarn`
- `p dump_tree(WN*)`
- `p dump_st(ST*)`
- `p dump_ty(TY_IDX)`
- `p dump_op (OP*)`
- `p dump_tn (TN*)`

# common data types

- WN\* // Whirl Node; common/com/wn\*
- ST\* // Symbol Table; common/com/symtab\*
- TY\_IDX // TYpe Index; common/com/symtab\*
- PREG // Pseudo-REGister; common/com/symtab\*
- TYPE\_ID | MTYPE // machine types; common/com/mtypes.h
- CODEREP\* // SSA expression; be/opt/opt\_htable.h
- STMTREP\* // SSA statement; be/opt/opt\_htable.h
- TN\* // Temporary Name; be/cg/tn.h
- OP\* // Operation; be/cg/op.h
- BB\* // Basic Block; be/cg/bb.h
- TOP // Target OPcode; targ\*/targ\_info/topcode.h

# Outline

- What it is
- Where it is
- How to build it
- How to use it
- How to debug it
- **How to change it**
- Future work

# Example: adding an intrinsic

- 4 kinds of intrinsics
  1. correspond to WHIRL instruction
  2. map to no-side-effect PTX
  3. have side effects
  4. use vectors

# Intrinsic 1 (WHIRL)

- example: f32 max
- in gccfe/gnu/builtins.def:

```
DEF_LIB_BUILTIN(BUILT_IN_FMAXF,  
                "__builtin_fmaxf",  
                BT_FN_FLOAT_FLOAT_FLOAT,  
                ATTR_NOTHROW_LIST)
```

# Intrinsic 1 (WHIRL)

- in gccfe/wfe\_expr.cxx:

```
case BUILT_IN_FMAXF:
```

```
    arg1 = TREE_VALUE (arglist);  
    arg2 = TREE_VALUE (TREE_CHAIN (arglist));  
    wn = WN_CreateExp2 (OPR_MAX, ret_mtype, MTYPE_V,  
        WFE_Expand_Expr (arg1), WFE_Expand_Expr(arg2) );  
    whirl_generated = TRUE;
```

- in .B file:

```
LOC 1 6      f = fmaxf(g,1.0f);  
F4F4LDID 0 <1,33,g> T<10,.predef_F4,4>  
F4CONST <1,35,0f3f800000>  
F4MAX  
F4STID 0 <1,32,f> T<10,.predef_F4,4>
```

# Intrinsic 1 (WHIRL)

- `be/cg/NVISA/expand.cxx::Expand_Max()` produces CG OP:  
[ 6] `TN64003 :- max.f32 TN64001 TN64002 ;`  
assigned registers:  
[ 6] `TN64003($f3) :- max.f32 TN64001($f1) TN64002($f2) ;`
- PTX:  
`max.f32 $f3, $f1, $f2;`
- TN == Temporary Name
  - can hold register, constant, or symbol names

# Intrinsic 2 (intrinsic\_op)

- pure with no side effects
- example: f32 sin
- common/com/wintrinsic.h: INTRN\_F4SIN
- common/com/intrn\_info.cxx:

```
{ /* F4SIN */  
    BYVAL, PURE, NO_SIDEEFFECTS, DOES_RETURN, NOT_ACTUAL,  
    CGINTRINSIC, IRETURN_F4, NULL, "SIN", "sinf"},
```

- gccfe/wfe\_expr.cxx:

```
case BUILT_IN_SINF:  
    iopc = INTRN_F4SIN;  
    intrinsic_op = TRUE;
```

# Intrinsic 2 (intrinsic\_op)

- **WHIRL:**

```
LOC 1 5    f = sinf(f);  
    F4F4LDID 0 <1,32,f> T<10,.,predef_F4,4>  
    F4PARAM 2 T<10,.,predef_F4,4> # by_value  
    F4INTRINSIC_OP 1 <251,SIN> 0  
    F4STID 0 <1,32,f> T<10,.,predef_F4,4>
```

- **be/cg/NVISA/expand.cxx:**

```
case INTRN_F4SIN:  
    Build_OP (TOP_sin_f32, result, op0, op1, ops);
```

# targ\_info

- common/targ\_info/isa/NVISA
- C++ files generate accessor files in targ\*/targ\_info/
- isa.cxx – add instruction name
- isa\_operands.cxx – describe operands
- isa\_print.cxx – how to print to .ptx file
- isa\_properties.cxx – e.g. TOP\_is\_load(t)

# Intrinsic 3 (intrinsic\_call)

- has side effects so don't optimize
- example: clock
- gccfe/wfe\_expr.cxx:

```
WN *wn = WN_Create_Intrinsic (OPC_I4INTRINSIC_CALL,  
    INTRN_CLOCK, 0, NULL);
```

- calls are statements
- return value in next statement
- preg = Pseudo-REGister

# Intrinsic 3 (intrinsic\_call)

- **WHIRL:**

```
LOC 12 26    c2 = clock(); // Read clock register
I4INTRINSIC_CALL <789,CLOCK> 0 # flags 0x0
I4I4LDID -1 <1,31,.preg_return_val> T<4,.predef_I4,4>
I4STID 34 <1,2,.preg_I4> T<4,.predef_I4,4> # <preg>
I4I4LDID 34 <1,2,.preg_I4> T<4,.predef_I4,4> # <preg>
I4STID 0 <2,5,c2> T<4,.predef_I4,4>
```

- **be/cg/NVISA/expand.cxx:**

```
case INTRN_CLOCK:
    call_iretult = PREG_To_TN (Int_Preg, First_Int_Preg_Return_Offset);
    Build_OP (TOP_mov_u32, call_iretult, Clock_TN(), ops);
    return call_iretult;
```

# Intrinsic 4 (asm)

- intrinsic uses vectors
- vectors not basic type in Open64 & GCC
- vectors look like structs
- builtins won't work, so use asm
- example: texfetch
- gccfe/wfe\_expr.cxx:

```
if (strcmp(name, "__utexfetchi1D") == 0) {  
    wn = emit_builtin_texfetch(exp,  
                               "tex.1d.v4.u32.s32", MTYPE_U4, MTYPE_I4);  
    asm_generated = TRUE;
```

# Outline

- What it is
- Where it is
- How to build it
- How to use it
- How to debug it
- How to change it
- **Future work**

# Future Work

- new hw features via intrinsics
- dwarf generation
- integrating with Open64 updates
- tune wopt to minimize register pressure
- unrolling
- using 16-bit instructions
- supporting calls
- analyze code to generate ideas