OpenACC Workers for AMD GCN

Julian Brown

March 2018

mentor embedded

mentor.com/embedded

Outline

- AMD GCN recap
- OpenACC recap
- Worker implementation:
  - Pass overview
  - Neutering
  - Broadcasting
- Worker reductions
- Future plans

Workgroups & Wavefronts

- A **workgroup** is equivalent to a CPU
  - Small amount of local memory (LDS)
- A **wavefront** is like a CPU thread:
  - Distinct **register set**
  - Distinct **stack**
  - **Asynchronous execution** wrt. other wavefronts
- Wavefronts can **synchronise** at barriers, workgroups cannot (easily)

OpenACC Execution Model

- Blocks of C/C++/Fortran code are **offloaded** to a GPU
- Loops within those blocks can be **distributed** over "abstract" parallelism levels:
  - The coarsest-grain level is **gangs**
  - Each gang is split into **workers**
  - Each worker is split into **vectors**
- At **each point** in an offloaded block, work may be distributed over gangs, workers and/or vectors
- If we are in **worker single** or **vector single** mode, it is important that side effects happen **only once**

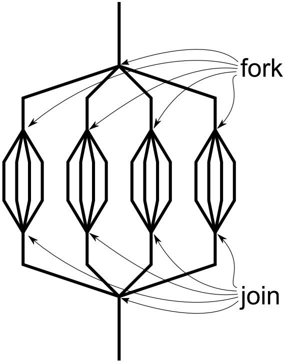
OpenACC on GCN

- For OpenACC, the following hardware features are used for parallelism:
  - "Gangs" correspond to **workgroups**
  - "Workers" correpond to **wavefronts**
  - "Vectors" use SIMD instructions
    - A vector lane is a **work item**
- Prior to this work, we were restricted to **one wavefront** per workgroup (a single worker)
- Parts of code derived from NVPTX implementation (but moved to the **middle-end**)
- This work concerns **OpenACC** only: OpenMP uses a different, more "CPU-like" scheme

Kernel Launch

- All workgroups/wavefronts ("threads") execute **the same code**, and run until completion
- HSA provides a "**3-dimensional**" model
  - $x * y * z$ work items (threads) are scheduled to run on the GPU
  - HSA is a cross-platform API. For GCN:
    - One of these dimensions maps to **SIMD vectors**
    - One dimension maps to **workgroups**
    - One dimension maps to **wavefronts** (i.e. workers)
- No provision for dynamically changing the number of threads
  - In particular, **all workers** run **all kernel code**
  - But, there are several ways of working around this

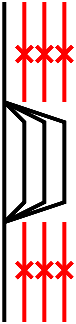
Middle-End Representation



- At the gimple level, "**fork**" and "**join**" primitives are used to demark partitioned loops
- For Nvidia PTX, these are rewritten in the backend to use a neutering/broadcast scheme
  - "**Simulating**" fork/join semantics
- For AMD GCN, we do a similar transformation **much earlier in compilation**

Worker Transformations

Transformations done at gimple level consist of:



- **Neutering**
  - A *control flow* transformation
  - Ensures that worker-single code executes on the **first wavefront** only
- **Broadcasting**
  - A *data flow* transformation
  - Ensures that **local state** changes on first wavefront are propagated to other (idle) wavefronts
  - Works on predicates used for **control flow**
    - Idle wavefronts "follow along" with the first

Pass Overview

1. **Split** basic blocks at fork/join boundaries
2. Par discovery: scan the function's **loop structure**
3. Populate single-mode bitmaps: record which basic blocks execute in **worker-single** or **vector-single** mode
4. Find **SSA names** which may need propagation (defined in worker-single mode)
5. Find uses of `VAR_DECL`s in worker-partitioned mode
6. Calculate set of local vars that may need propagating after each worker-single block:
   6.1 Those that are **assigned directly**
   6.2 Those that may be modified by a **write through a pointer**
7. **Transform** worker-single mode blocks using above data

Block Splitting

- **Split blocks** that contain forks or joins
  - Parallelism level changes on **edges**, not within blocks
- Some stmts are put in **singleton** blocks in **fully-partitioned** mode:
  - Control flow (`GIMPLE_RETURN`, `GIMPLE_COND`, `GIMPLE_SWITCH`, `GIMPLE_CALL`)
  - Assignments with `COMPONENT_REF`, `BIT_FIELD_REF`, `ARRAY_REF` lhs

Transforming Conditions

Condition splitting

```
// [...]
if (a > b) goto blk1; else goto blk2;
```

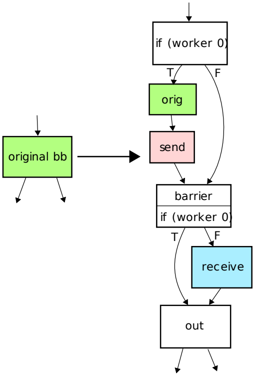
to...

```
// Worker zero executes this statement:
pred = (a > b);
// Block split, next stmt executed by
// all workers:
if (pred != 0) goto blk1; else goto blk2;
```

Transforming Function Calls

- Two types of function calls to consider in worker-single mode:
  - **"Normal" calls** – to maths library routines, etc.
  - OpenACC **routines**
- The former can be left as-is, and be called from **worker zero** only
- The latter may contain worker-partitioned loops, so call from **all workers** (including "idle" ones)
  - Worker-single code within the function undergoes the **same transformation**

Neutering (1)



- A single bb is transformed into a **graph** of blocks:
  - **Predicate block** inserted at top
  - Original block executed for a **single worker** only
  - Thread-local state is **broadcast** via LDS
  - Other workers **receive** local state changes after sync barrier

Neutering (2)

In pseudocode:

```
static __lds oacc_ws_data_s_1 oblk;

if ((iblk = __builtin_oacc_single_copy_start(&oblk))
     == NULL) {
  // Do stuff
  oblk.x = x;
  oblk.y = y;
  __builtin_oacc_single_copy_end (&oblk);
}
__builtin_oacc_worker_barrier ();
if (iblk) {
  x = iblk->x;
  y = iblk->y;
}
```

Broadcasting (1)

We want to propagate **thread-local** state:

- Register contents
- The stack

But not:

- Global memory, including mapped buffers

Broadcasting (2)

To simulate "fork" via broadcasting, gimple entities we need to process are:

- **SSA names** (≈ machine registers)
- Local **scalar variables** (≈ stack slots)
- Local **aggregates** (≈ stack slots also)
- Pointer **indirection**

Unlike NVPTX (or a real "fork"), we do not know the "real" machine registers, nor the final contents/layout of the stack.

SSA Broadcast (1)

Maintaining SSA form

```
x_5 = <something>;
```

to...

```
if (worker == 0)
  x_5 = <something>;
else
  x_6 = 0;
x_7 = PHI (x_5, x_6);
```

- SSA names may **have definitions** in a worker-single block
- After neutering, definitions may **no longer dominate** uses
- We must invent a definition for the **idle edge** too
- Insert a **Φ-node** at the convergence point

SSA Broadcast (2)

- An SSA name (defined in worker-single mode) can be used...
  - ...in the **current block** only
  - ...in **other worker-single blocks** only, or
  - in **worker-partitioned** mode
- For the last case, we must broadcast by:
  - Copying **to LDS** after active block
  - Copying **from LDS** in other block
- We don't need to broadcast via LDS if **all uses** are in **worker-single mode**
- We don't need a Φ-node if all uses are **within the current block**

Local Variable Broadcast

- Addressable local scalar (non-aggregate) variables are **not rewritten** to SSA form
- Broadcast any variables that are **written** in the current **worker-single** block and are **read** in **worker-partitioned** mode
  - Probably pessimistic, but safe and flow-insensitive

Local Aggregates

- Local aggregates (arrays, structures) are **not** broadcast
- Instead, gimple stmts modifying elements/fields of local aggregates are forced into **fully-partitioned** mode
- The operation is done **redundantly** by all workers
- The RHS of the gimple assignment will be a scalar, thus **subject to broadcasting**

Writes Through Pointers

- Writes through pointers may affect **any local variable** (that has its address taken)
- Use GCC's **points-to analysis** to determine the set of potentially-affected variables
  - Lets us ask, for a given pointer indirection and variable, "might this pointer point to this variable?"
  - At `-O0`, falls back to "yes" – any local variable may be modified
- Broadcast **any such variable** which is used in **worker-partitioned** mode
- Done on a **per-block** basis, for any block with a write through a pointer

Worker Reductions

- OpenACC reductions use a set of gimple builtin functions interleaved with fork, join, etc. markers
- Generally works well with the GCN workers implementation
- Some trouble with reductions to **reference** variables in Fortran (e.g. function arguments):
  - **Address taken** in **worker-single** mode
  - Address broadcast, then **dereferenced** in **worker-partitioned** mode
  - All wavefronts access **worker zero's stack** – oops!
- Solved by **rewriting** reference reductions to use local non-reference copies of variables (a patch by Cesar, slightly modified)

Future Work

- Neutering for **single-entry, single-exit** (SESE) regions instead of a single basic block at a time
  - The code is mostly there already (from NVPTX), but not wired up yet
- Removal of **duplicate barriers**
- Increase number of **concurrent workers** (tune SGPR/VGPR usage, LDS usage)
- Try sharing the new gimple workers code with NVPTX too
  - Potential speed or maintenance benefits
  - Vector single/vector partitioned mode handling would need more work

Thank You!