Status of High Level Loop Optimizations in GCC

Richard Günther

Jul. 9th, 2012

Not so high-level loop optimizations

Work on innermost loops or at least do not consider a loop nest as whole.

Not so high-level loop optimizations

- loop header copying
- final value replacement

- complete loop peeling
- loop invariant motion and store sinking
- loop unswitching
- loop distribution and pattern detection
- loop if-conversion
- loop vectorization
- predictive commoning
- loop induction variable optimization
- loop array prefetching
- loop unrolling and peeling

Not so high-level loop optimizations

- loop header copying
- final value replacement

- complete loop peeling
- loop invariant motion and store sinking
- loop unswitching
- loop distribution and pattern detection
- loop if-conversion
- loop vectorization
- predictive commoning
- loop induction variable optimization
- loop array prefetching
- loop unrolling and peeling

High level loop optimizations

Work on loop nests and change the shape or order of the loop tree.
Memory hierarchy optimization.

High level loop optimizations

Work on loop nests and change the shape or order of the loop tree.
Memory hierarchy optimization.

High level loop optimizations

- loop parallelization
- loop strip mining
- loop blocking
- loop interchange

- unroll and jam
- working interchange
- better heuristics for complete loop nest unrolling

High level loop optimizations

- loop parallelization
- loop strip mining
- loop blocking
- loop interchange

- unroll and jam
- working interchange
- better heuristics for complete loop nest unrolling

Preserving Loop Structure

- Alongside the regular
- Track individual loops throughout the entire compilation
- Attach information
- Makes some CFG transforms hard
- Destroy loop form
- Inhibit CFG transform

Preserving Loop Structure

- Alongside the regular
- Track individual loops throughout the entire compilation
- Attach information
- Makes some CFG transforms hard
- Destroy loop form
- Inhibit CFG transform

Pass managing

- When to do (highlevel) loop transforms?
- Cost model issues
- Too many tiny scalar optimization passes in random order
- Inflexible static pass pipeline

Pass managing

- When to do (highlevel) loop transforms?
- Cost model issues
- Too many tiny scalar optimization passes in random order
- Inflexible static pass pipeline

GRAPHITE

GRAPHITE.

GRAPHITE pros

- Almost all high level loop optimizations implemented in the GRAPHITE framework
- The polyhedral model is well researched and other compilers use it
- No phase-ordering issues(?)
- GRAPHITE has been updated to work with ISL instead of PPL and with a recent CLooG version
- Possibly using shared infrastructure between GCC and LLVM (Polly)

GRAPHITE cons

- The GCC side of GRAPHITE is still considered unmaintained due to lack of ressources
- Polyhedral optimizations are slow and memory-hungry
- Discourages work on non-GRAPHITE high level loop optimizers?

Alternatives

- Re-surrect the old interchange code?
- Middle-end array expressions?
- Other suitable representation for high-level loop optimizations?
- Simply throw away GRAPHITE first?
- Volunteers?

Alternatives

- Re-surrect the old interchange code?
- Middle-end array expressions?
- Other suitable representation for high-level loop optimizations?
- Simply throw away GRAPHITE first?

- Volunteers?

Discussion

.

Discussion.

.