



Expanding the **IMPACT** Toolbox

ADVANCED COMPILER TECHNOLOGY



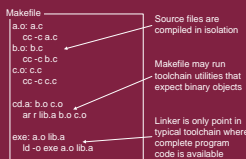
Christopher Rodrigues, Sara Sadeghi, Shane Ryoo, Robert Kidd, Sain-Zee Ueng, Wen-mei Hwu

Abstract

The current, labor-intensive paradigm of designing parallel systems is a deterrent to fully utilizing the computing power of contemporary multiprocessing systems. To address this issue, we propose a new, interactive, visualization- and interface-driven framework that eases developer efforts in obtaining multi-threaded performance. This framework will provide flexible tools to analyze and transform code correctly and quickly, allowing developers to rapidly iterate through various designs and improve time to final product.

Advanced Optimization on Real Code

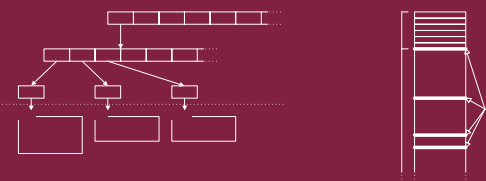
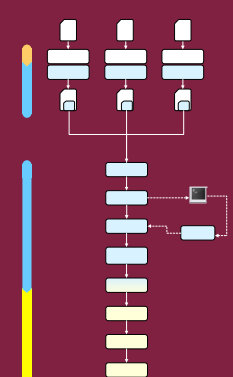
The build system (makefile) used for many programs makes whole program optimization difficult



To perform advanced optimization on real code, must preserve IR at compile time, continue compilation at link time

Many programs are too large to keep the full representation in memory. IMPACT's IR uses Keys as a unique name for symbols and to locate symbols in the symbol table. The Key is valid whether in memory or on disk

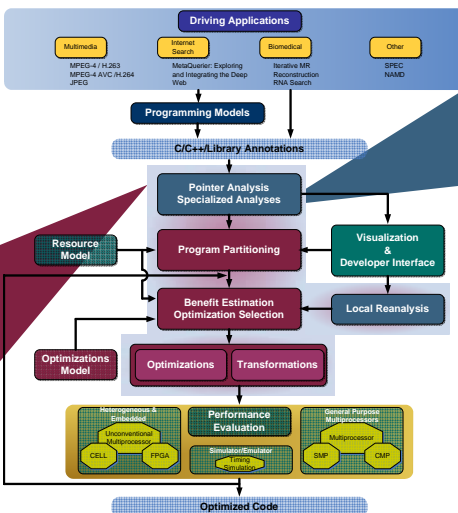
(a) The symbol table is constructed as a two level index. The top level contains one entry for each source file (IR dump) in the program. The second level is a per-file index that contains each symbol defined in a source file. The data structure pointed to by the second index level contains all information necessary to read IR from disk. After IR has been read from disk, it is pointed to by the Symbol structure. (b) The IR dump for each source file contains the representation of the second index level and the IR for the file. Sentinels between each IR symbol allow the file to be indexed for random access.



Deep Analysis

Find the set of possible variable values, data flow, and execution paths in a program

- Pointer Analysis
- Flow-sensitive Pointer Refinement
- Value Constraint
- Memory Dataflow



Target Capabilities

```

/* Potentially parallel code */
if (config->pan) {
  for (n=0; n<config->nsamp; n++) {
    t1 = buf_l[n];
    t2 = buf_r[n];
    buf_l[n] = 0.9*t1 + 0.1*t2;
    buf_r[n] = 0.1*t2 + 0.9*t1;
  }
}

/* initialization routine */
cfg->nsamp = 576;
cfg->pan = balance ? 0 : 1;
return cfg;

/* main routine */
cfg = &default_t_cfg;
if (...) cfg = new_cfg(bal);
main_loop(cfg);
  
```

Analysis

Points-to Relations



Pointer aliasing: Are config, buf_l, buf_r independent objects?

Value Constraints

t1: unknown config->pan: 0 or 1
t2: unknown config->nsamp: 576

Value constraints: Are pan, nsamp known from other code? In other words: Will the loop execute? How many times?

Memory Dataflow

Datflow	Dependencies	Operations
1 3 5 ...		1 t1 = buf_l[0]
2 4 6 ...		2 t2 = buf_r[0]
		3 buf_l[0] = ...
		4 buf_r[0] = ...
		5 t1 = buf_l[1]
		6 t2 = buf_r[1]

Memory dataflow: What memory locations are read and written by the code?

Addressing real-world codes

Previously developed analyses for parallelization are imprecise or inapplicable when confronted with realistic C programs

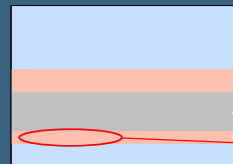
- Thousands of lines of code
- Recursion
- Dynamic allocation
- Variable size arrays
- Interprocedural control flow
- Run-time configuration parameters

Our approach is to find targeted and scalable techniques that can work on a large range of useful applications

- Summarization of large code segments
- Apply expensive, local analyses on demand
- Layer analyses to build on previous analyses' results
- Attack recurring difficult analysis problems with targeted schemes

Specialized analysis: Cycle sensitivity

To parallelize the loops within MPEG-4's VopMotionCompensate, the compiler or developer must identify and represent the relationship between recon and prev. However, previous "combined approaches" are not scalable/practical.



We use a partially flow-sensitive and context-sensitive analysis to narrow the set of objects pointed to by the variables, then analyze the flow pattern to determine the no-alias region for the two variables.