Dynamic Binary Translation for Generation of Cycle Accurate Architecture Simulators

Institut für Computersprachen
Technische Universtät Wien
Austria

Andreas Fellnhofer   Andreas Krall   David Riegler

Part of this work was supported by OnDemand Microelectronics and the Christian Doppler Forschungsgesellschaft

Overview

Previous Projects

- incremental compilers
- static assembly language instrumentation
- Molecule (ATOM clone)
- STonX
- CACAO
- bintrans
- reverse compilation (DSP VLIW to C)
- compiled cycle accurate instruction set simulation
- iboy

STonX

- AtariST on X windows
- generated 68k instruction set emulator
- work on binary translation unfinished

CACAO

- JIT-only Java Virtual Machine
- ultra-fast basic compiler
- recompilation with optimizations
- on-stack replacement
- deoptimization when assumptions become invalid

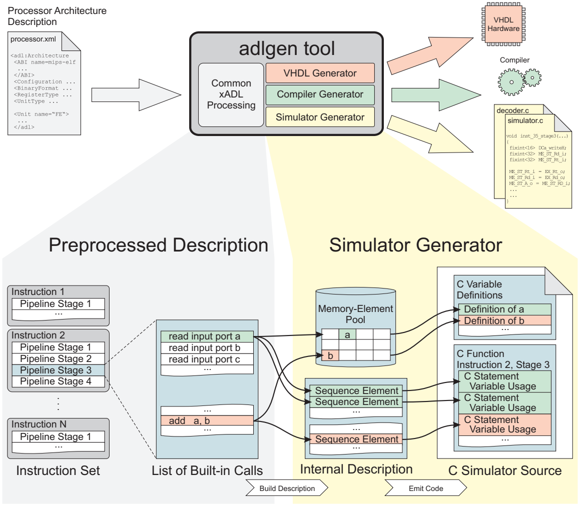
bintrans

- generator for user mode binary translators
- LISP like source and target architecture specification
- direct translation of blocks/traces without intermediate representation
- hybrid fixed register mapping and local register allocation
- local register liveness analysis with global propagation between different runs
- 1.8 to 2.5 overhead compared to native code

iboy

- gameboy emulator for iPod
- full system level cycle accurate emulation
- uses only dynamic binary translation
- self modifying code leads to recompilation of basic block
- template based generated compiler
- local flag constant propagation and liveness analysis
- ROM/RAM code caches

Overview



```
<adl:Architecture
 <ABI name=mips-elf
  ...
 </ABI>
 <Configuration ...
 <BinaryFormat ...
 <RegisterType ...
 <UnitType ...

 <Unit name="FE">
  ...
 </adl>
```

```
void inst_35_stage3(...)
{
 fixint<16> DCa_writeH;
 fixint<32> ME_ST_Rd_i;
 fixint<32> ME_ST_Rt_i;

 ME_ST_Rt_i = EX_Rt_o;
 ME_ST_Rd_i = EX_Rd_o;
 ME_ST_A_o = ME_ST_RD_i;
 ...
 ...
}
```

Architecture Description Language

- mostly structural
- xml syntax
- graphical user interface available
- no redundant information
- MIPS R2000 specification is about 1000 lines

Architecture Description Example

```
<Operation name="addu" syntax="op3_s" >
 <Syntax syntax="op3_s" token="op" value="addu" />
 <Body>
  <add a="Rs_i" b="Rt_i" d="Rd_o" o="overflow" c="carry"/>
 </Body>
</Operation>
```

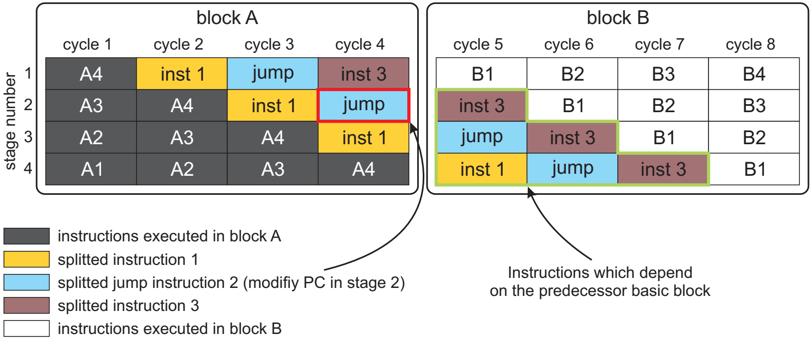
Simulator Basics

- generated mixed interpreting/translating simulator
- translation of blocks and traces
- common IR for interpreter and translator
- backend is LLVM just-in-time compiler
- own and LLVM optimizations used

Differences to Standard Binary Translation

- cycle accurate simulator
- full system simulation
- simulation of in-order pipelined architectures
- instructions cross basic block borders

Overlapping Instructions

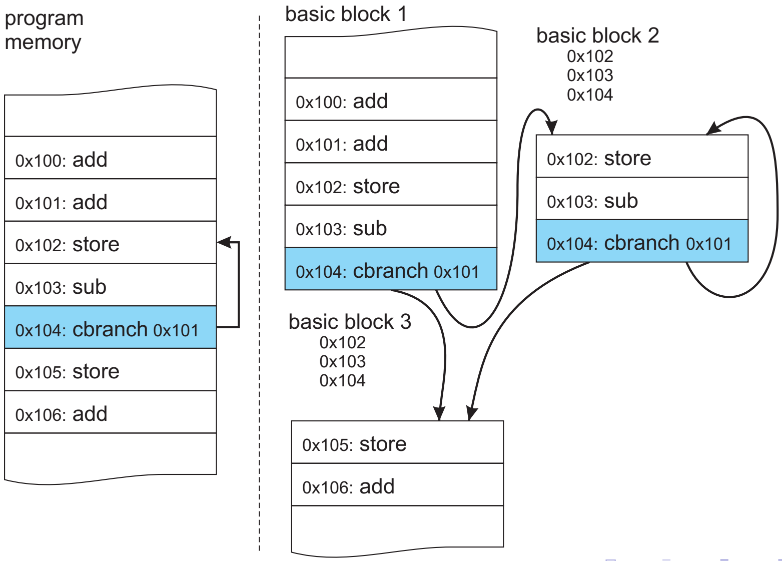


stage	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6	cycle 7	cycle 8
1	A4	inst 1	jump	inst 3	B1	B2	B3	B4
2	A3	A4	inst 1	jump	inst 3	B1	B2	B3
3	A2	A3	A4	inst 1	jump	inst 3	B1	B2
4	A1	A2	A3	A4	inst 1	jump	inst 3	B1

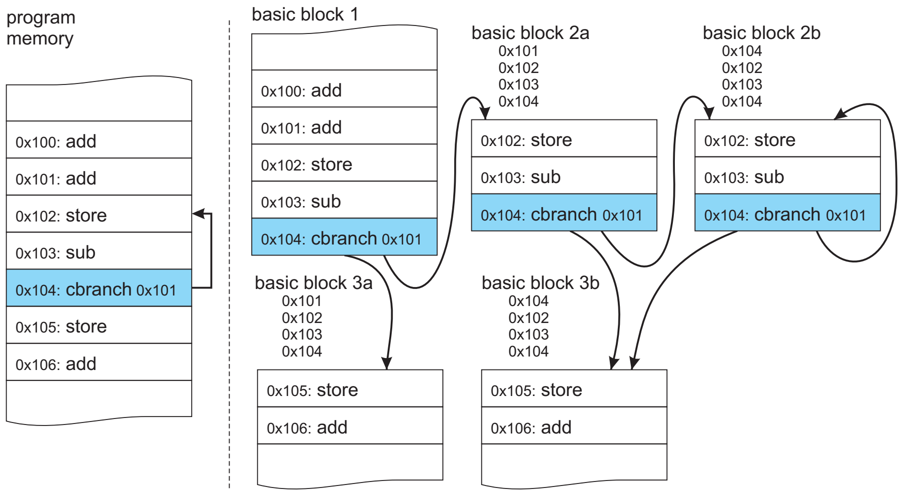
⬛	instructions executed in block A
🟨	splitted instruction 1
🟦	splitted jump instruction 2 (modifiy PC in stage 2)
🟫	splitted instruction 3
⬜	instructions executed in block B

Instructions which depend on the predecessor basic block

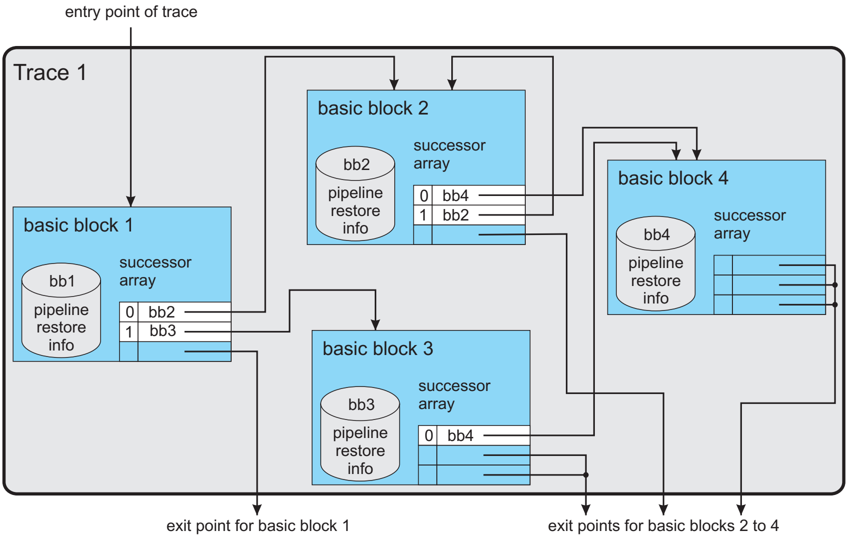
Similar Predecessor Blocks



Basic Block Duplication



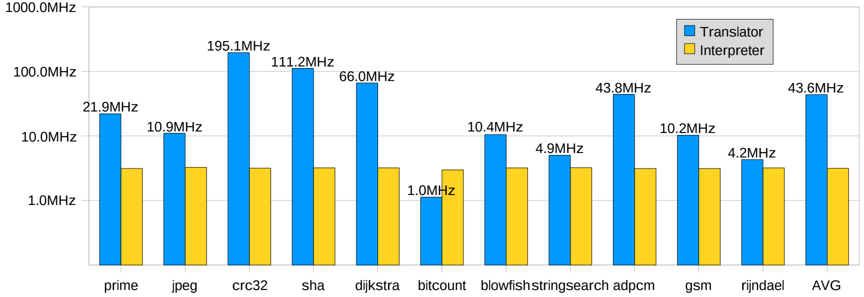
Trace Formation



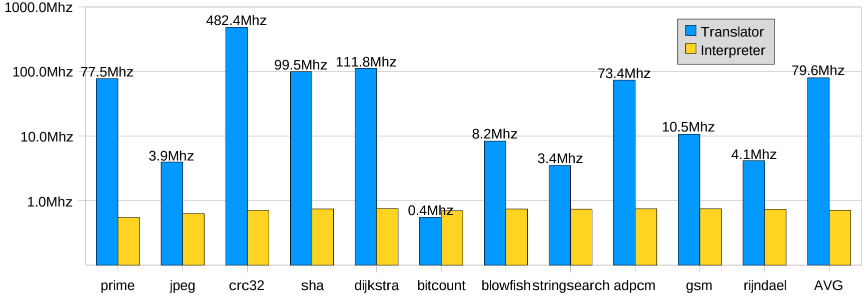
Optimizations

- LLVM optimizations (e.g. constant propagation, dead store elimination)
- local copy of global values
- linking of basic blocks
- constant forward optimization
- LLVM JIT very slow (mostly instruction selection)

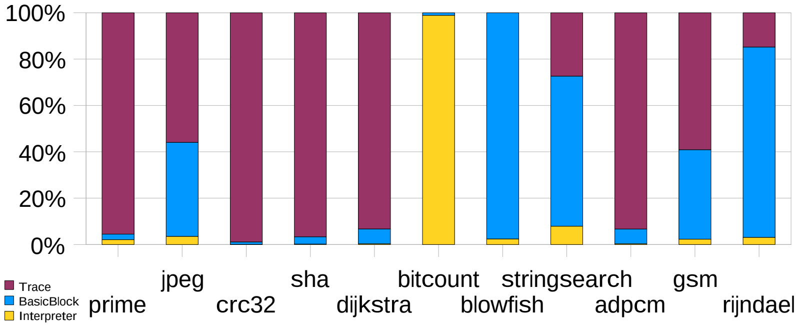
Simulation Speed MIPS



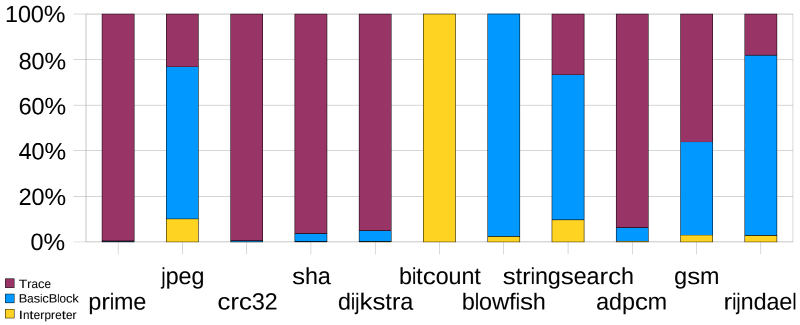
Simulation Speed CHILI



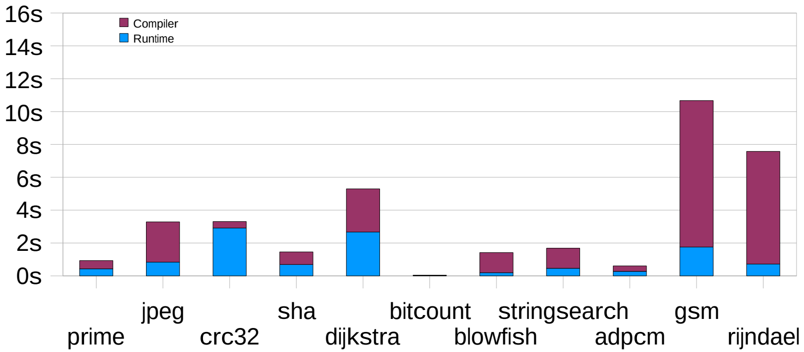
Breakdown of Simulation Cycles MIPS



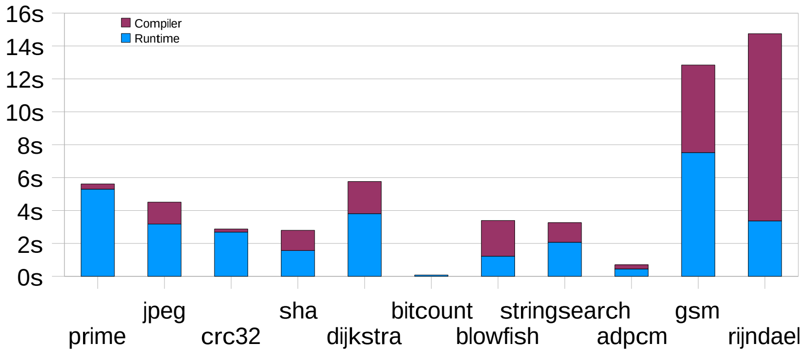
Breakdown of Simulation Cycles CHILI



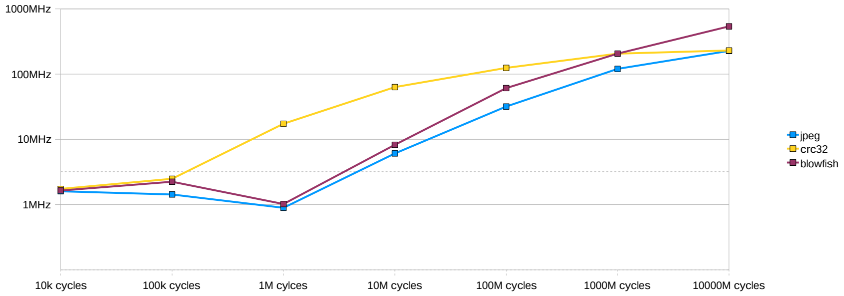
Compile and overall Run Time MIPS



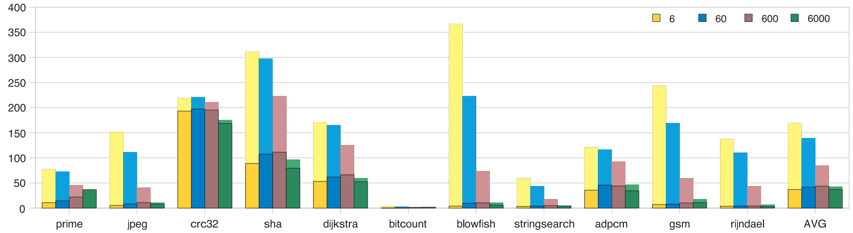
Compile and overall Run Time CHILI



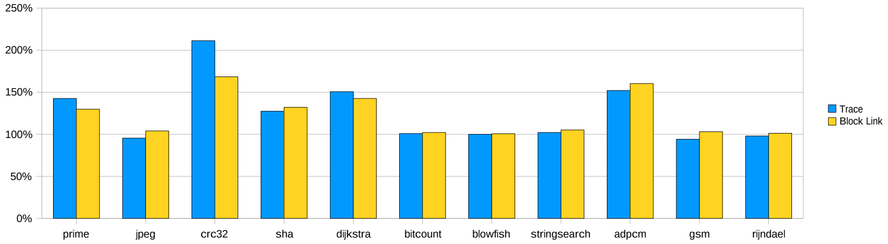
Performance with increasing Simulation Time CHILI



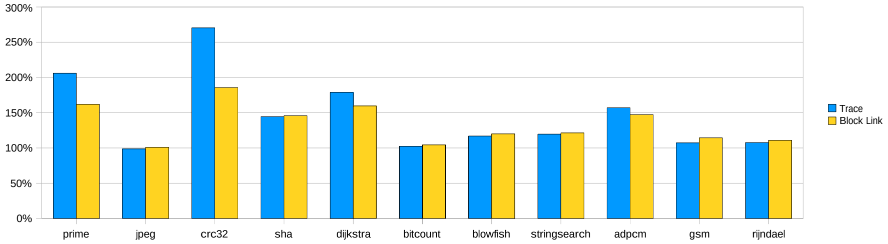
Simulation Speed with different Compilation Thresholds



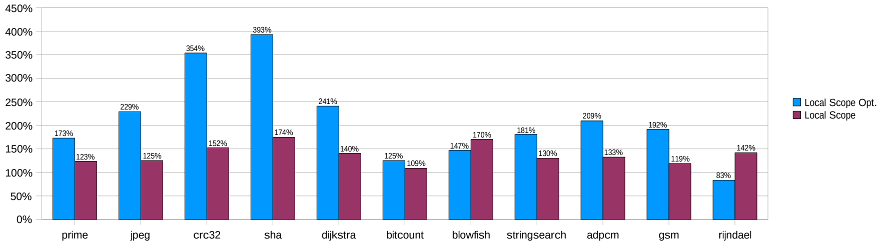
Optimized Block Linkage MIPS



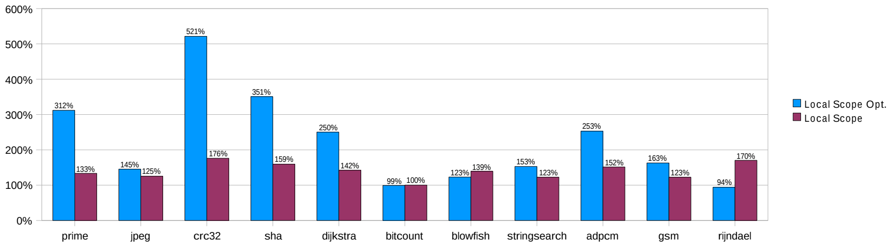
Optimized Block Linkage CHILI



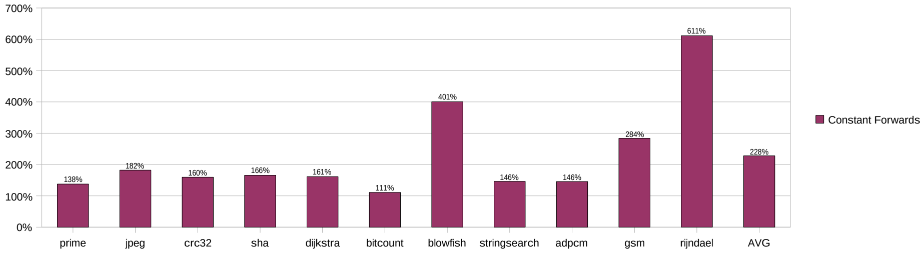
Local Copy of Global Values MIPS



Local Copy of Global Values CHILI



Forwarding Optimization MIPS



Conclusion

- cycle accurate simulation rises additional problems
- binary translation is efficient
- LLVM JIT is slow