Open Research Compiler (ORC): A Compiler Infrastructure for Research

Co-organizers:
Chu-cheow Lim, Roy Ju
(MTL, Intel Labs)
Junchao Zhang, Li Chen, Xiaobing Feng, Chengyong Wu
(ICT, CAS)

Presented at the 17th International Workshop on Languages and Compilers for Parallel Computing
(LCPC'04)

West Lafayette, Indiana
September 22, 2004

Outline

- Overview of ORC
- ORC Infrastructure and Components
- ORC Aid tools
- ORC/Open64 Activities
- ORC and AutoPar
- Future Plan

Overview of ORC

ORC

- <u>Achieved objective</u>: be a leading open source IPF (IA-64) compiler infrastructure to the compiler and architecture research community
- <u>Requirements: </u>robustness, timely availability, flexibility and performance
- Download site: http://ipf-orc.sourceforge.net/
- Mailing lists: ipf-orc-support@lists.sourceforge.net and open64-devel@lists.sourceforge.net

*IPF for Itanium Processor Family in this presentation

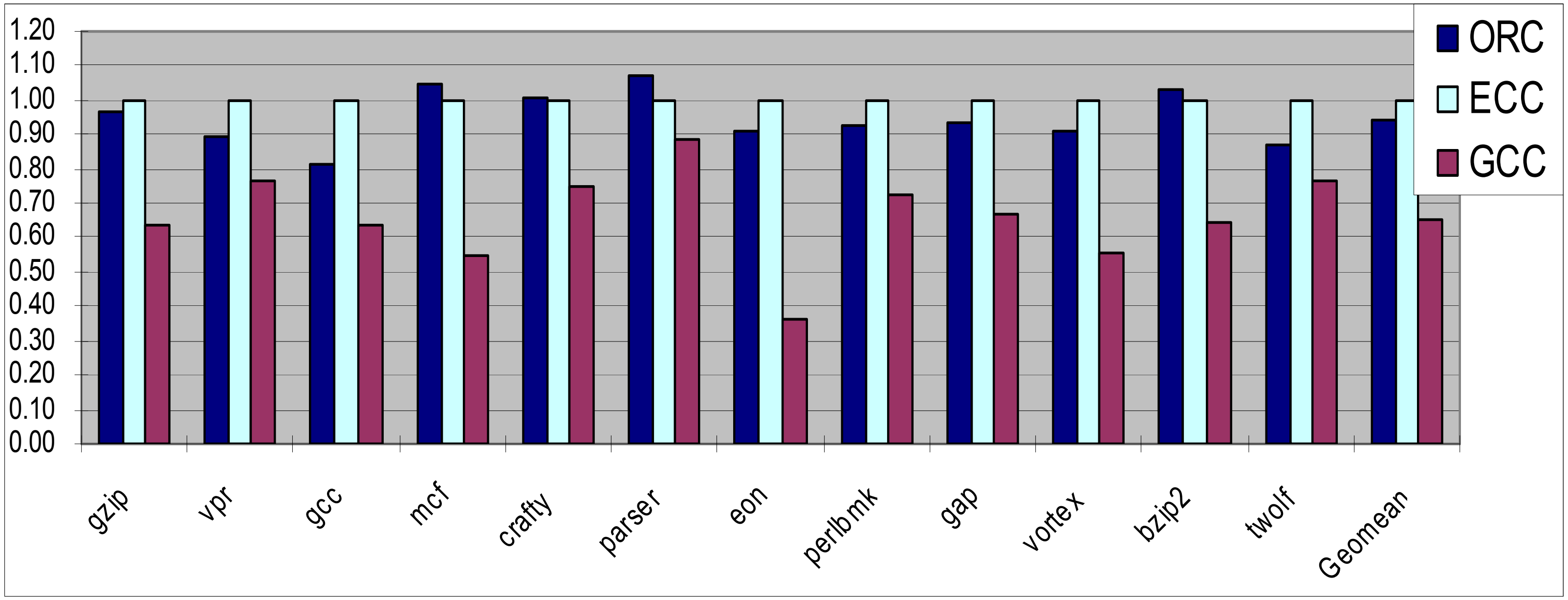
The ORC Project

- ORC development efforts started in Q4 2000
- Based on the Pro64 open source compiler released by SGI
  - Retargeted from the MIPSPro product compiler
- Joint efforts among
  - Programming Systems Lab (PSL), MTL, Intel
  - Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS)
  - Intel China Research Center (ICRC), MTL, Intel
- Core engineering team: 15 - 20 people
- Received contributions from the Open64 community and various users

ORC Releases

- ORC1.0 released in Jan’02
  - ORC basic infrastructure and key features in IPF optimizations
- ORC1.1 released in Jul’02
  - Inter-procedural optimization, WHIRL feedback profile and edge profile were fully functional.
  - Numerous performance enhancements
- ORC2.0 released in Jan’03
  - Key performance release
  - Performance features, including C++ and memory optimization
- ORC2.1 released in Jul’03
  - Performance features for Itanium2
- ~14,500 downloads from the first release till now
- ORC2.2 planned in Q4 2004

ORC2.1 Performance on Itanium2/Linux



- Testing environment:
  - 4-way 900MHz Itanium2 , 16K L1 DCache,16K L1 ICache, 256K L2 Cache, 3M L3 Cache, 1G Mem, RedHat AS2.1
  - Compiled with SPEC base options for Ecc 7.0 and –O3 for Gcc 3.1
- ORC 2.1 at 5% from ECC 7.0 and 30% ahead of Gcc 3.1

ORC Infrastructure and Components

Intermediate Representations (IR) of ORC

- WHIRL (Winning Hierarchical Intermediate Representation Language), an AST-based IR with references to symbol table
  - Bridges semantic gap from source languages to machine instructions
  - Same IR for multiple languages and multiple targets
  - Interfaces between compiler components
    - Well defined input and output
  - Medium for IR-based optimizations
- Well documented and released by SGI

WHIRL

- WHIRL has 5 levels of abstraction from very high to very low
  - Each level semantically the same
  - Less information at lower level
  - Allow optimizations to be performed at the most appropriate representations
- Continuous lowering of IR through each component
  - Call ***lowerer*** to translate to lower level
- Supporting components to
  - Minimize variations in IR form
  - Avoid duplication of optimization/analysis functionalities

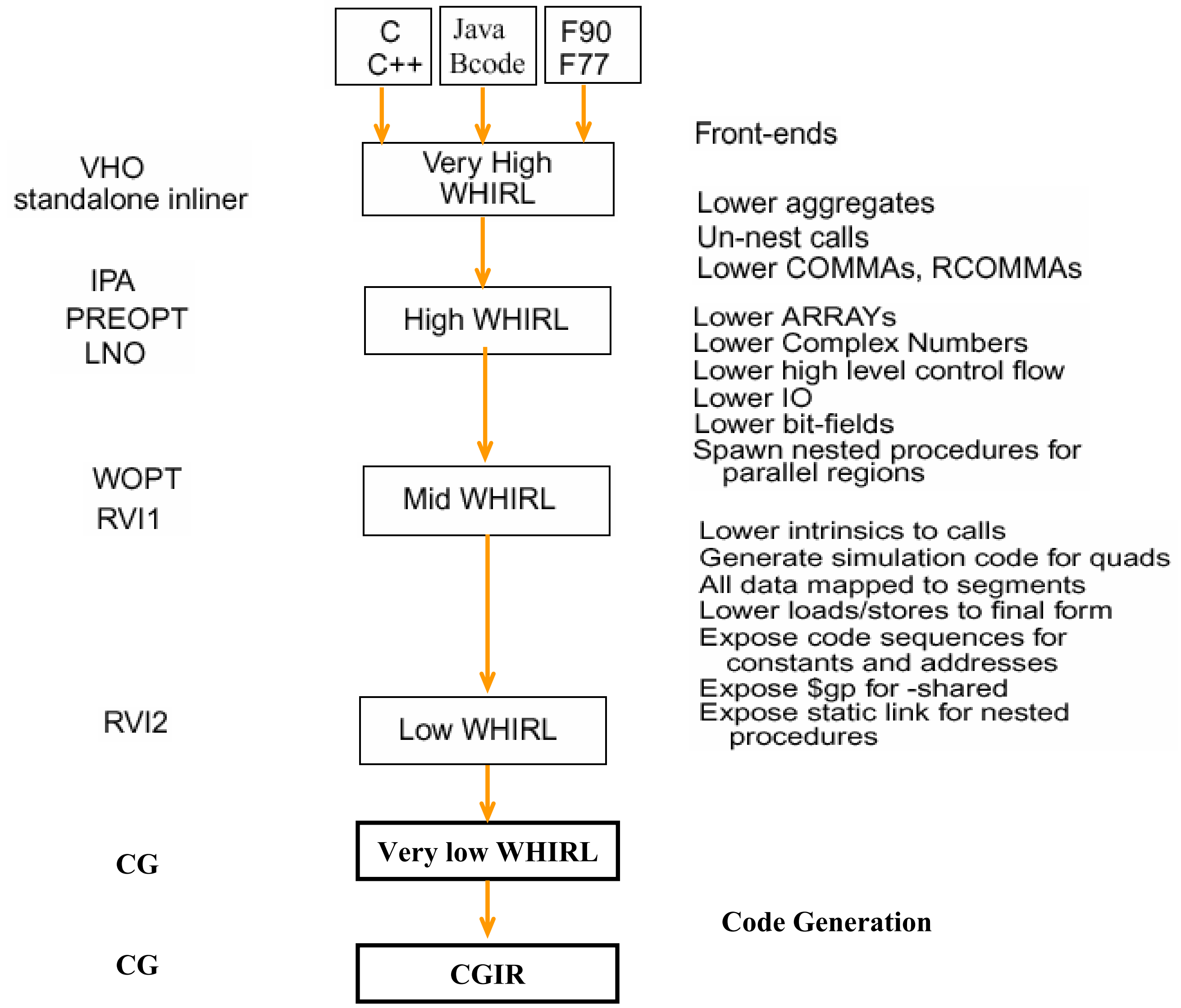
Symbol Table and CGIR

- Symbol table
  - Also Documented and released by SGI
- Code generation IR (CGIR)
  - Conventional register-based IR for load/store architecture
  - Flags on basic blocks, operations, operands

Tools for WHIRL

- ***ir_b2a***
  - Dumps IR in pre/post order form
- ***whirl2c, whirl2f***
  - Raise high level and up WHIRL back up to C/Fortran
  - May not be re-compilable if input is low-WHIRL
- Numerous verifiers and self-checkers throughout compiler
  - WHIRL consistency
  - Symbol table and WHIRL consistency

Flow of IR



Front-ends

Lower aggregates
Un-nest calls
Lower COMMAs, RCOMMAs

Lower ARRAYs
Lower Complex Numbers
Lower high level control flow
Lower IO
Lower bit-fields
Spawn nested procedures for parallel regions

Lower intrinsics to calls
Generate simulation code for quads
All data mapped to segments
Lower loads/stores to final form
Expose code sequences for constants and addresses
Expose $gp for -shared
Expose static link for nested procedures

Code Generation

Front Ends of ORC

- C front end based on gcc2.96
- C++ front end based on g++2.96
- Fortran90/95 front end from MIPSPro
  - Can't bootstrap itself for including files written in Fortran90

Optimizations Based on WHIRL

- Mainly inherited from Pro64
- Inter-procedural analysis and optimizations (IPA)
  - Call tree construction, alias analysis, array section analysis, mod/ref summary
  - Fully integrated inlining, cloning, dead function and variable elimination, etc.
- Loop-nest optimizations (LNO)
  - Locality optimization, parallelization, loop distribution, unimodular transformations, array privatization, OpenMP, …

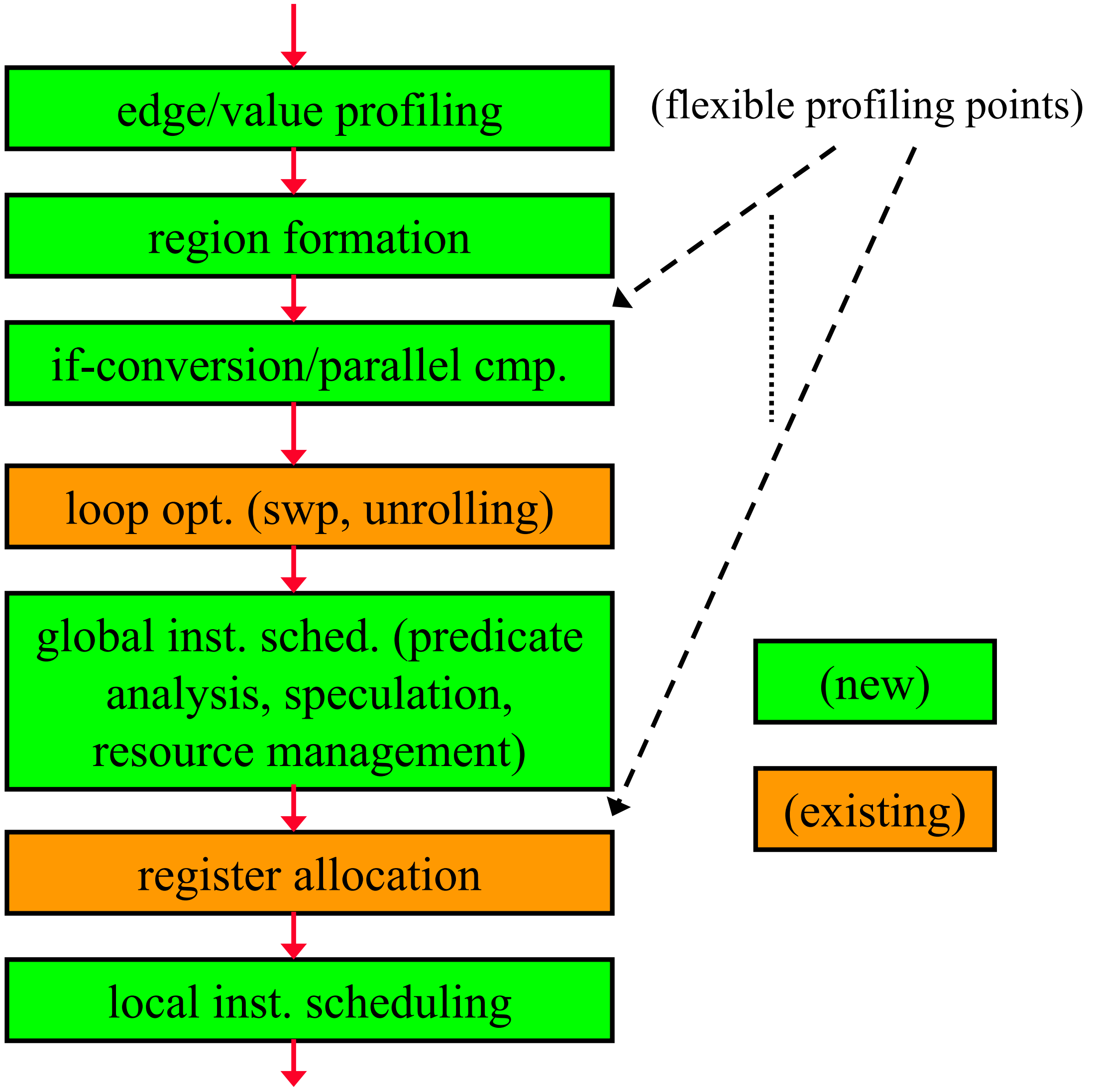
Optimizations Based on WHIRL (cont.)

- Scalar global optimizations (WOPT)
  - SSA is unifying technology
  - Use only SSA as program representation
  - All traditional global optimizations implemented
    - Partial redundancy elimination (PRE)
    - Induction variable recognition
    - Strength reduction
    - Load/store-PRE
    - Copy propagation, …
  - Every optimization preserves SSA form
- See Pro64 tutorial by G. Gao et al at PACT’00 (http://www.cs.ualberta.ca/~amaral/Pro64/index.html) and publications from MIPSPro posted by SGI

What’s new in CG?

- CG has been largely redesigned from Pro64
- Research infrastructure features:
  - Region-based compilation
  - Rich profiling support
  - Parameterized machine descriptions
- IPF optimizations:
  - If-conversion and predicate analysis
  - Control and data speculation with recovery code generation
  - Global instruction scheduling with resource management
- Other enhancements

Major Phase Ordering in CG


edge/value profiling
region formation
if-conversion/parallel cmp.
loop opt. (swp, unrolling)
global inst. sched. (predicate analysis, speculation, resource management)
register allocation
local inst. scheduling
(flexible profiling points)
(new)
(existing)

Region-based Compilation

- Motivations:
  - To form profitable scopes for optimizations
  - To control compilation time and space
- Region
  - A connected subset of CFG, single-entry-multiple-exit, acyclic, hierarchical, more general than hyperblocks, treegion, etc
- Region formation algorithm decoupled from the region structure
- Optimization-guiding attributes at each region
- Being used to do multi-threading research
- See *Y. Liu et al, A Region-Based Compilation Infrastructure, INTERACT'03*

Profiling Support

- Edge profiling at WHIRL in Pro64 remained and extended
- New profiling support added in CG to allow various instrumentation points
- Types of profiling
  - Edge profiling, value profiling, memory profiling, …
- Important for limit study or collecting program statistics
- User model
  - Instrumentation and feedback annotation at same point of compilation phase
  - Later phases maintain valid feedback information through propagation and verification

Parameterized Machine Model and Micro-scheduler

- Motivations
  - To centralize the (micro-)architectural details and facilitate the study of hardware/compiler co-design
- Machine model
  - Reads in the (micro-)architecture parameters
  - Automatically generates machine description tables
- Micro-scheduler
  - Manages resource constraints, models inst. dispersal rules, interacts with the high-level inst. scheduler and reorders inst. within a cycle
  - Uses a finite state automata (FSA) to model the resource constraints
    - Each state represents occupied FU's
    - State transition triggered by incoming scheduling candidates
- See *D. Y. Chen et al, Efficient Resource Management during Instruction Scheduling for the EPIC Architecture, PACT'03*

If-conversion and Predicate Analysis

- If-conversion
  - Converts control flow to predicated instructions
  - Iteratively detect patterns for if-conversion candidates within regions
  - Utilizes parallel compare instructions to reduce control dependence height
- Predicate analysis
  - Analyze relations among predicates and control flow and store them in predicate relation database (PRDB)
  - Query interfaces to PRDB: disjoint, subset/superset, complementary, sum, difference, probability, …
  - PRDB can be deleted and recomputed as wish without affecting correctness
- No coupling between the if-conversion and predicate analysis

Global Instruction Scheduling

- Performs on the scope of SEME regions
- Cycle scheduling with priority function based on frequency-weighted path lengths
- Full resource management by interacting with micro-scheduler
- Modularizes the legality and profitability testing
- Includes
  - Safe speculation across basic blocks
  - Control and data speculation
  - Code motion with compensation code
  - Partial ready code motion
  - Motion of predicated instructions
  - …

Control and Data Speculation

- Speculative dependence edges added on DAG
- Selection of speculation candidates driven by scheduling priority function
- For a speculated load, insert *chk* and add DAG edges to ensure recoverability
- Recovery code generation decoupled from scheduling phase
- Starting from the speculative load, follow flow and output dependences to re-identify speculated instructions
- GRA to properly color registers in recovery blocks
- See *R. Ju et al, A Unified Compiler Framework for Control and Data Speculation, PACT'00*

Other Phases in CG

- Software pipeline (SWP)
  - Lifetime sensitive modulo-scheduling for both while and do loops
- Register allocation (RA)
  - A hybrid of Chaitin/Briggs's optimistic graph coloring and Fred. Chow's priority-based approach
- Extended Block Optimization (EBO)
  - Simple peephole optimizations on extended blocks, including constant propagation, redundant expression or dead expression elimination
- Control Flow Optimization (CFO)
  - Optimizations on CFG, like deleting unreachable code, collapsing empty goto's

Design for Debugability (DFD) and Testability (DFT)

- DFD and DFT built-in from start
- Can build with extra validity checks
- Trivial effort to add new options
- Simple option specification used to
    - Substitute components known to be good
    - Enable/disable full components or specific optimizations
    - Enable/disable optimizations for specific functions, blocks, operations
    - Invoke alternative heuristics
    - Trace IRs flowing through each phase and activities taken in each phase

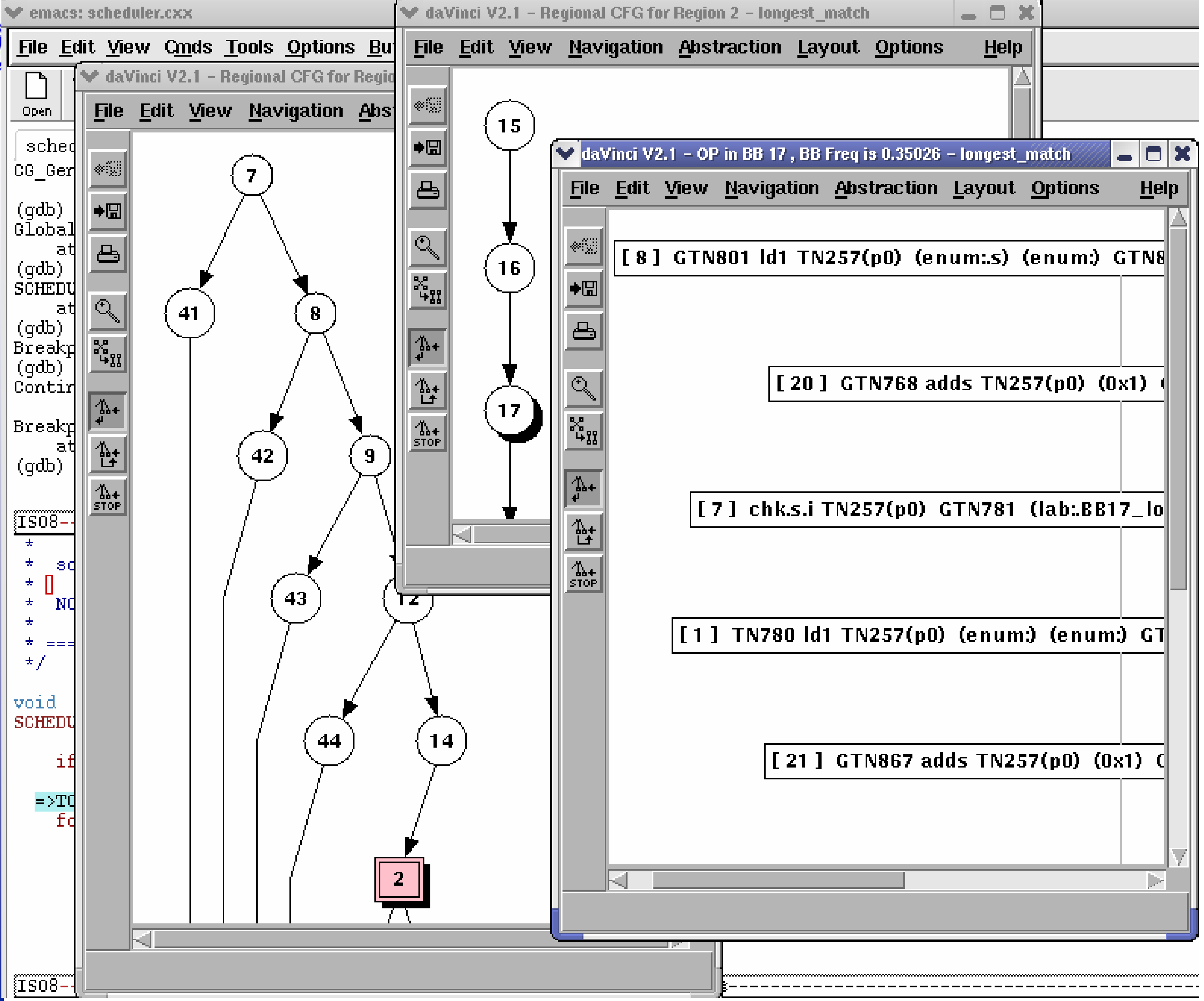
ORC Aid Tools

ORC Aid Tools

- Auto test scripts
  - Update and build the compiler
  - Test benchmarks, collect results, email test reports
  - Show checkin and performance history
  - Log and timestamp all activities during testing
- Triage tool
  - Auto locate the .o file, procedure, opt. phase, basic block, even the instruction that triggers a runtime bug by binary search

ORC Aid Tools (cont.)

- Visualization Tools
  - Based on daVinci, an X-Window visualization tool from University of Bremen
  - Communicate with daVinci in gdb, visualize graph-based data structures
    - Global and regional CFG, region tree
    - Regional and local (basic block) dependence graph
    - Partition graph
  - Show the CFG of assembly code

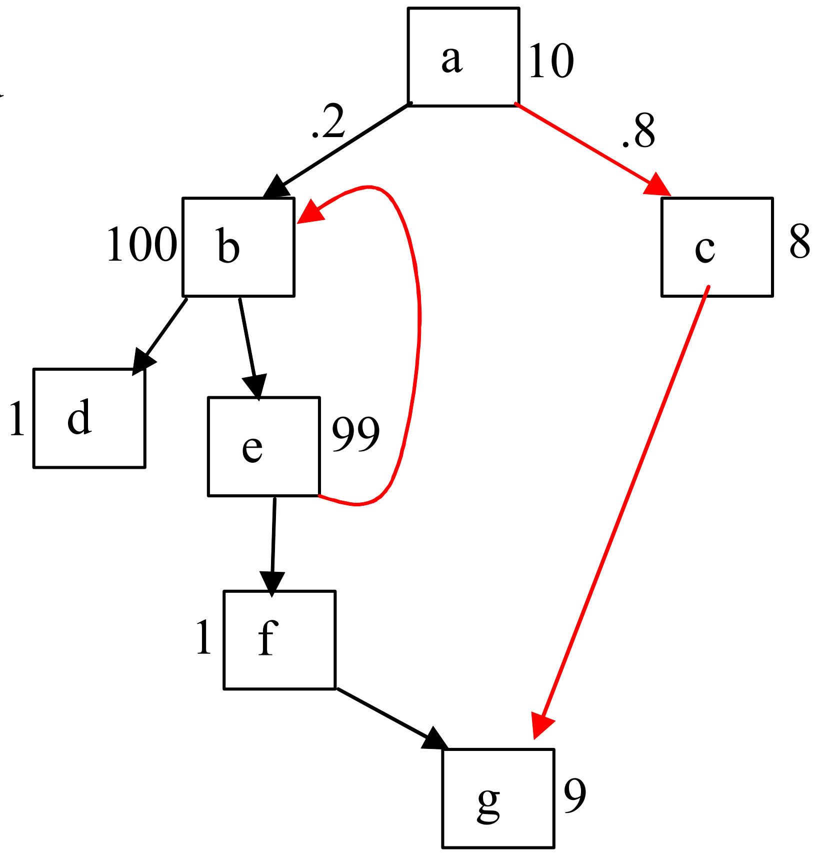

emacs: scheduler.cxx
File Edit View Cmds Tools Options Bu
Open
daVinci V2.1 – Regional CFG for Regio
File Edit View Navigation Abs
7
41
8
42
9
43
12
44
14
2
daVinci V2.1 – Regional CFG for Region 2 – longest_match
File Edit View Navigation Abstraction Layout Options Help
15
16
17
STOP
daVinci V2.1 – OP in BB 17 , BB Freq is 0.35026 – longest_match
File Edit View Navigation Abstraction Layout Options Help

```
 sched
CG_Ger

(gdb)
Global
   at
(gdb)
SCHEDU
   at
(gdb)
Breakp
(gdb)
Contir

Breakp
   at
(gdb)

IS08--
 *
 *  sc
 * []
 *  NC
 *
 * ===
 */

void
SCHEDU

   if

  =>TC
   fc

IS08--
```

```
[ 8 ] GTN801 ld1 TN257(p0) (enum:.s) (enum:) GTN8
[ 20 ] GTN768 adds TN257(p0) (0x1) (
[ 7 ] chk.s.i TN257(p0) GTN781 (lab:.BB17_lo
[ 1 ] TN780 ld1 TN257(p0) (enum:) (enum:) GT
[ 21 ] GTN867 adds TN257(p0) (0x1) (
```

ORC Aid Tools (cont.)

- Cycle counting tool
  - Counts the cycles of pre-selected hot functions by scanning assembly files
  - Used to find performance degraded functions
- Hot path enumeration tool
  - Finding compiler performance defects through analyzing assembly code is a tedious work
  - Analyzing assembly code on hot paths of hot functions is more efficient



ORC/Open64 Activities

ORC/Open64 in Academia

- ICT, Chinese Academy of Sciences
  - Port ORC to Godson, an MIPS compatible 64-bit general purpose processor designed by ICT
  - Use ORC to do research on automatic parallelizing compilation on distributed memory machines
- Tsinghua Univ., China
  - Software pipelining (SWP)
    - Research on decision heuristics of SWP and cache latency aware modulo-scheduling (LAMS)
  - OpenMP (http://sourceforge.net/projects/orc-openmp)
    - Explore thread-level parallelism
    - Make ORC compliant to OpenMP
    - ORC-2.0-OpenMP-1.1 released in May’04

ORC/Open64 in Academia (cont.)

- Univ. of Houston
  - Use Open64 as an infrastructure for OpenMP research
- Univ. of Delaware (Prof. G. Gao)
  - Low power/energy research on loop transformation and SWP
  - Open64-based Kylin compiler (kcc) for Intel Xscale processor
- Univ. of Minnesota (Profs P. Yew and W. Hsu)
  - Use ORC as an instrumentation and profiling tool to study alias, dependence, thread-level parallelism for speculative multithreaded architectures
- LBNL/U. C. Berkeley
  - Use Open64 as a UPC-to-C translator
  - UPC is a parallel extension to C for scientific computing
- Rice University
  - Source-to-source transformation of Fortran 90 programs with extensions

ORC/Open64 in Academia (cont.)

- University of Alberta (Prof. J. N. Amaral)
  - ORC/Open64 as class projects on machine SSA, pointer-based prefetching
  - Later phase SSA representation, profile-based partial inlining, multi-alloc placement
- Georgia Institute of Technology (Prof. Krishna Palem)
  - Use ORC in compile time memory optimizations, like data remapping, load dependence graphs, cache sensitive scheduling, static Markovian-based data prefetching and design space exploration
- Universiteit Gent, Belgium
  - Reuse distance-based cache hint selection, Euro-Par’02
- …(other universities)

ORC in Industry

- Intel
  - Shangri-La project
    - A network application programming environment based on ORC, includes
      - A domain specific high level language (Baker)
      - A pipeline compiler to partition Baker programs to multiple heterogeneous processors
      - A code generator for Intel IXP network processors
      - A runtime adaptation system
    - Joint efforts among
      - CTL, PSL, ICRC of Intel
      - ICT, CAS
      - U.T. Austin
  - Speculative Multi-Threading (SpMT)
    - Exploit thread-level parallelism by partitioning single threaded apps into potentially independent threads
    - Region-based optimizations intended to support multithreading
  - JIT leverages the ORC micro-scheduler

Open64 in Industry

- Tensilica
- STMicroelectronics
- PathScale
- …

ORC and AutoPar

Background

- What is AutoPar?
  - A source to source parallelizing compiler developed by ICT since 1990s
- Goals of AutoPar
  - Raise the automation level of parallelization
  - Pursue scalable performance
- What have been done in AutoPar?
  - Automatic data and computation decomposition
  - Hybrid parallelism exploitation, synchronization optimization
  - Partially replicated computation partition, inter-node pipelining
  - Communication optimization

What Does ORC Bring to AutoPar?

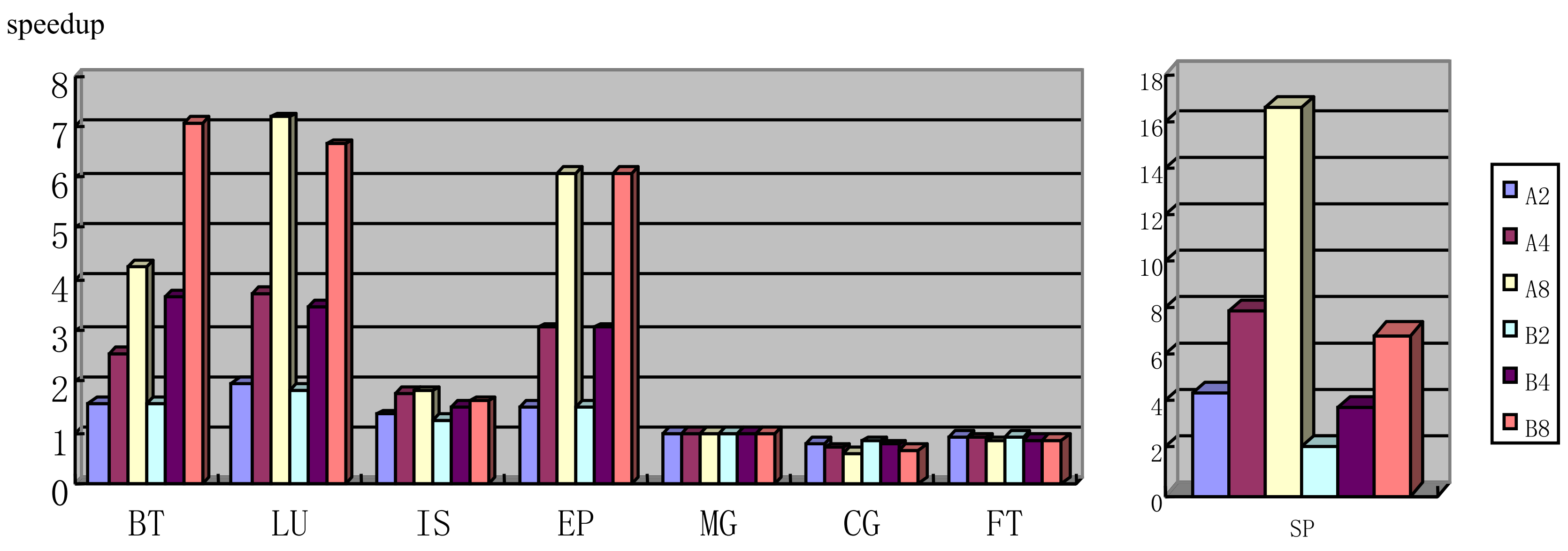
- Strong data dependence analysis
  - Extensive and flexible dependence test methods
    - GCD, *SOE(SVPC →Acyclic→ FM elimination), …*
  - Accurate mod/ref analysis
    - Array section analysis in IPA (disabled in ORC)
- Array data flow analysis within loops (disabled in ORC)
  - Each section is a list of bounded convex regions
- Parallelizing and padding phase
- Rich profiling support

Current Status and Future of AutoPar

- Have engrafted AutoPar with ORC by passing the required information from ORC to AutoPar as directives in source code
  - Raised the automation level when parallelizing complex loop structures
    - More inter-node pipelining, and parallelization across call sites
  - More aggressive mod/ref analysis for arrays
    - Near-optimal inter-procedural program decomposition
- Preliminary results
  - Parallelizing is nearly full-automatic except with few data distribution directives
    - A better inter-procedural computation partitioning framework
    - A better data distribution framework
- To improve AutoPar with efficient profiling support
- To build a parallelizing optimizing compiler infrastructure based on the techniques from AutoPar and ORC

Performance of AutoPar

- Test environment
  - DeepComp6800 (No.14 of Top500 in Nov.2003)
  - 197 1.3GHz Itanium2 (4way SMP), 2G Mem, RedHat AS2.1
  - Network connection: QSnet of Quadrics
  - Native compiler efc7.0, -O2;
- Only use 1 data distribution directive in NPB1.0 (appbt)



Future Plan

Future Plan

- Release of ORC2.2
  - ~ Q4 of 2004
- Features of ORC2.2
  - Upgrade the front end from gcc2.96 to gcc3.3
    - Compatible with ISO C++98 standard, binutils2.14 and glibc2.3.2
    - WHIRL front end (WFE) and gcc front end are both revised to cooperate with each other
  - A lot of whirl2c and whirl2f bug fixes
- Beyond ORC2.2
  - Integrate the parallel compiler, AutoPar, with ORC to explore the coarse-grained parallelism
- ORC will continue to be supported

Acknowledgements

- Institute of Computing Technology, Chinese Academy of Sciences
- Programming Systems Lab, MTL, Intel
- Intel China Research Center, MTL, Intel
- Pro64 developers
- ORC/Open64 users