

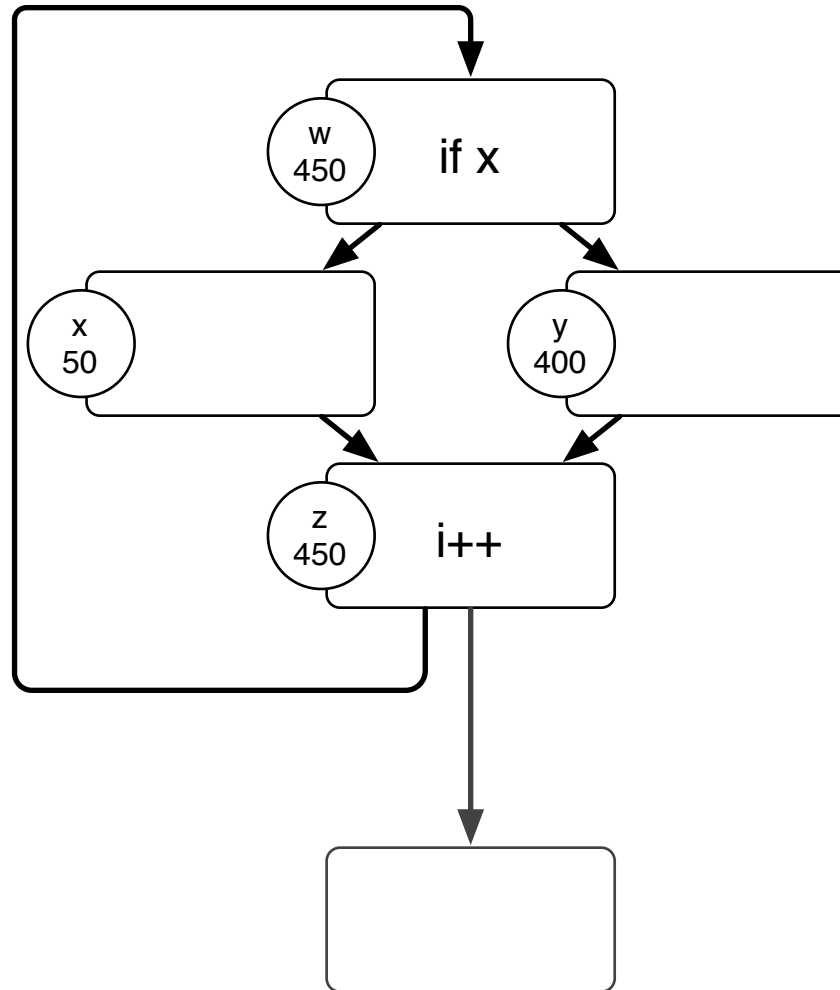
Improved Superblock Optimization in GCC

Robert Kidd
GCC Summit
June 29, 2006

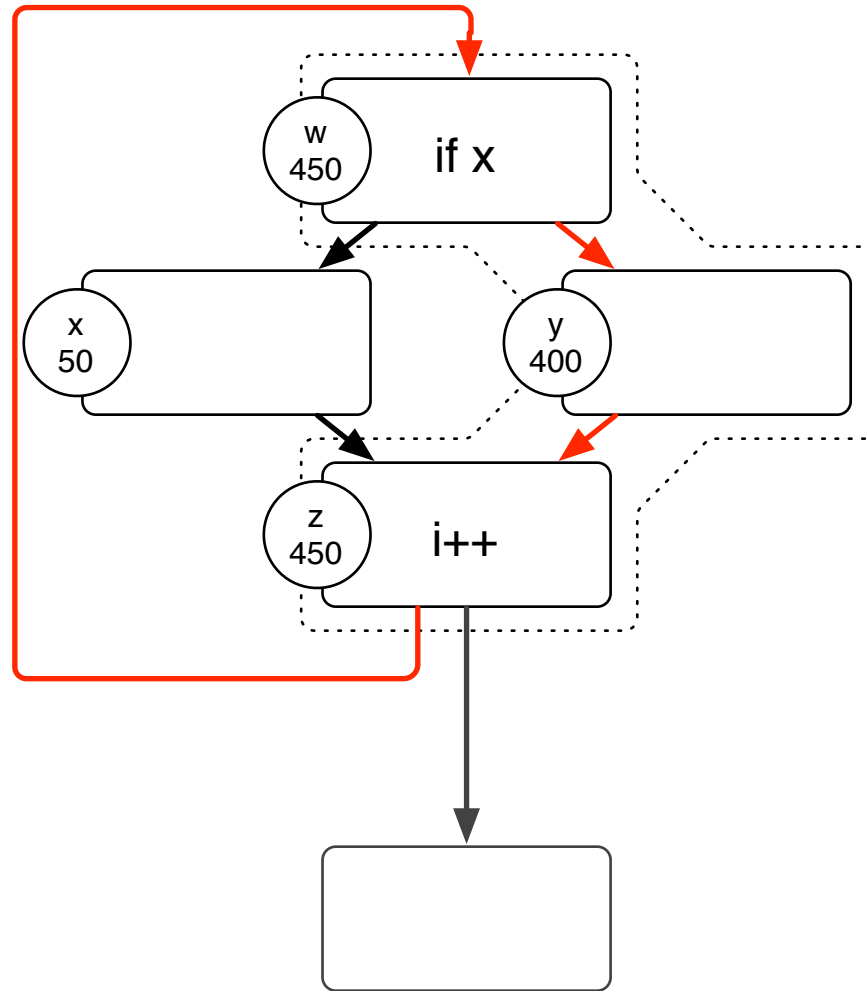
Introduction and history

- Superblock scheduling originally developed in the IMPACT compiler to increase ILP on VLIW processors
- Research in Superblock-based compilation led to the development of the structural compilation model used in OpenIMPACT
- This project aims to develop structural techniques in GCC to improve performance on IA64 and other processors

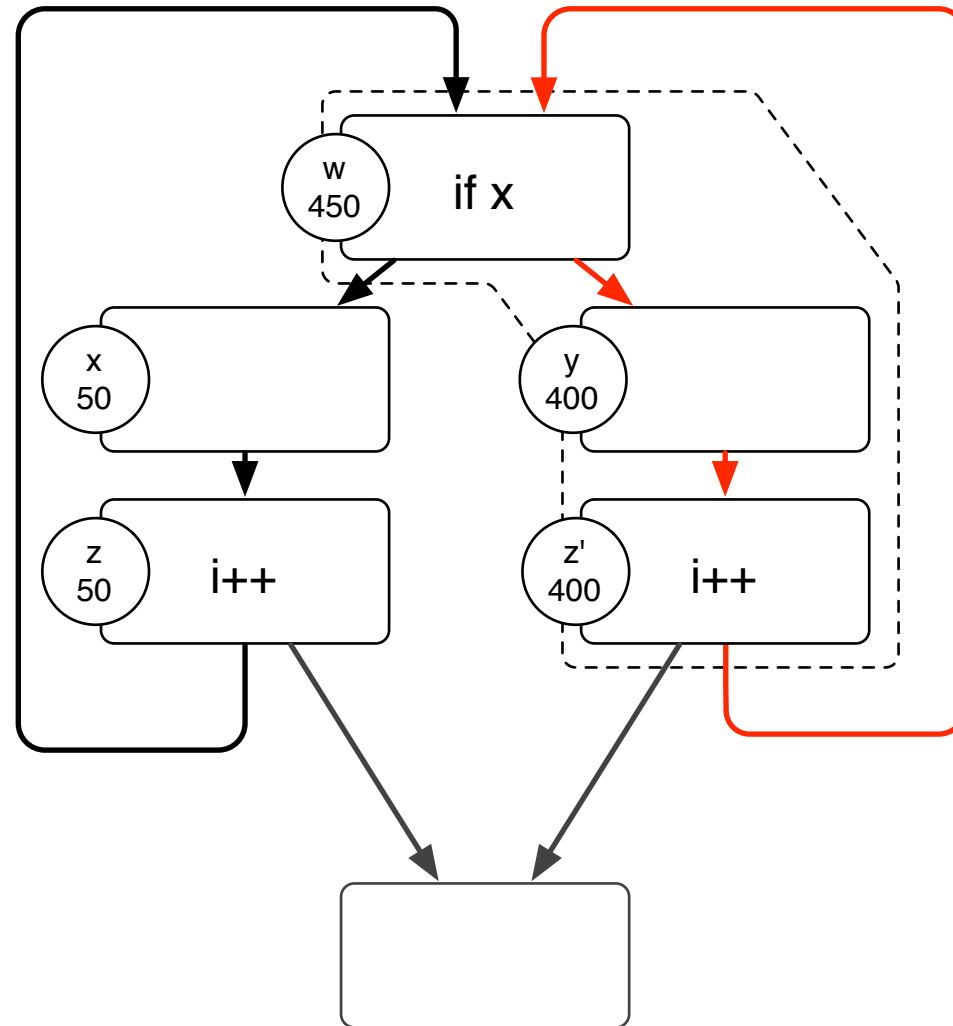
Introduction to Superblocks



Introduction to Superblocks



Introduction to Superblocks



Superblock scheduling

- Eliminating side entrances through tail duplication allows most of the benefit of trace scheduling with less bookkeeping
 - Effectively generates compensation code before moving instructions
- Code expansion offset by compacting the useful code, increasing ILP
- Generalizing this anticipatory code expansion leads to the idea of structural compilation, developed in OpenIMPACT

Structural compilation

- Multiple Superblock-like code expanding transforms followed by high-level optimization
- Aggressively expand code, simplifying control flow along the expected path of execution
- High level optimizations specialize duplicated blocks along the expected path
- Longer straight-line sequences allow greater ILP

Structural compilation in OpenIMPACT

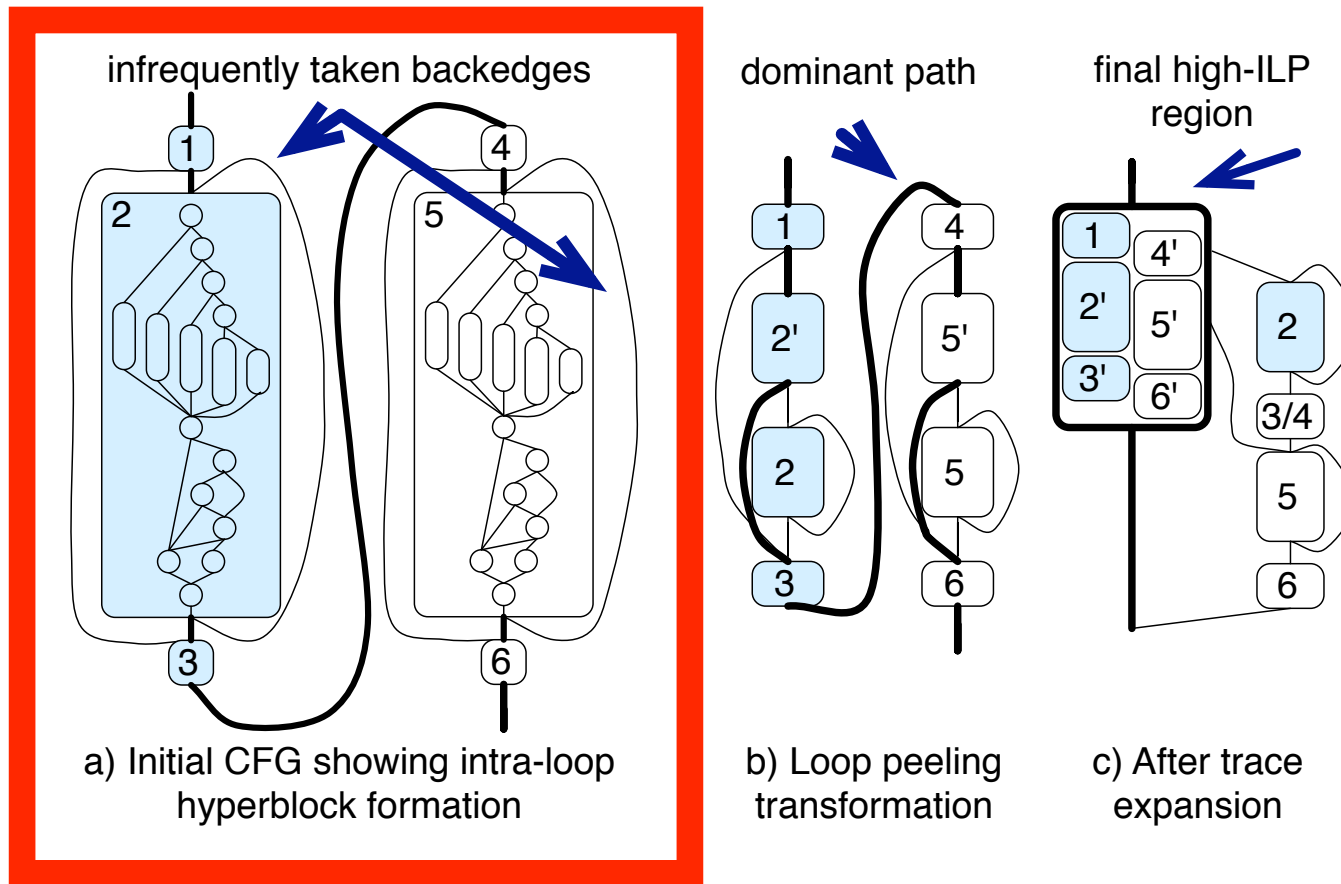
- On Itanium 2, structural compilation achieves a 13% increase in performance across SPECint2000
- Code size increases by approximately 1.5x
- Structural techniques tend to cluster frequently executed code, mitigating the code expansion
 - Higher ILP increases number of useful instructions per fetch
 - Straight line code increases I-cache fetch efficiency
- Structural techniques should also benefit processors that are less sensitive to scheduling

186.crafty in OpenIMPACT

- A combination of structural techniques transform the program's CFG
- Traditional optimizations specialize the duplicated code
- Inside the expected path of execution, more ILP is developed in a shorter schedule
- ~9% improvement

Combining structural transformations

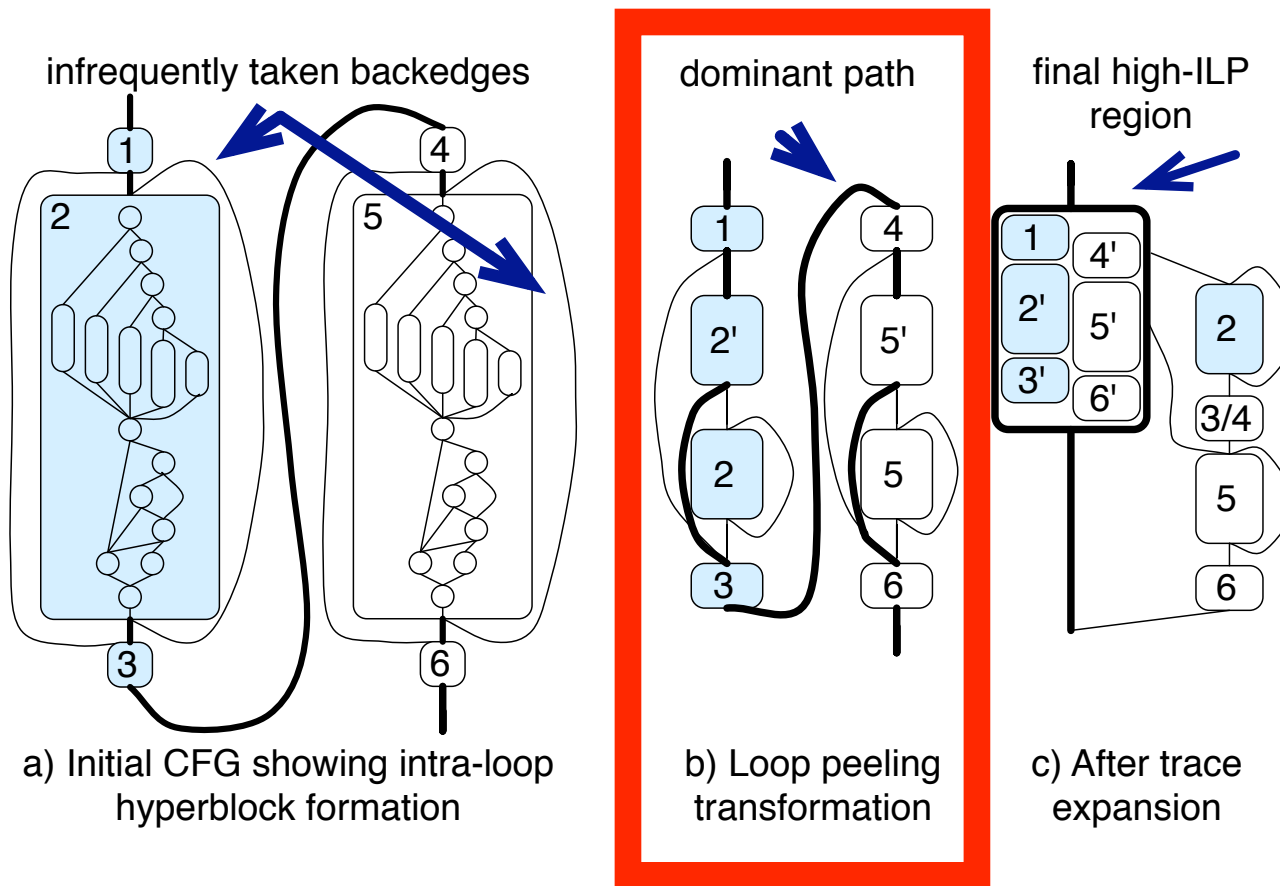
- After Hyperblock formation, serial, low trip count loops limit ILP



Example from `I86.crafty:Evaluate()` [SiasISCA04]

Combining structural transformations

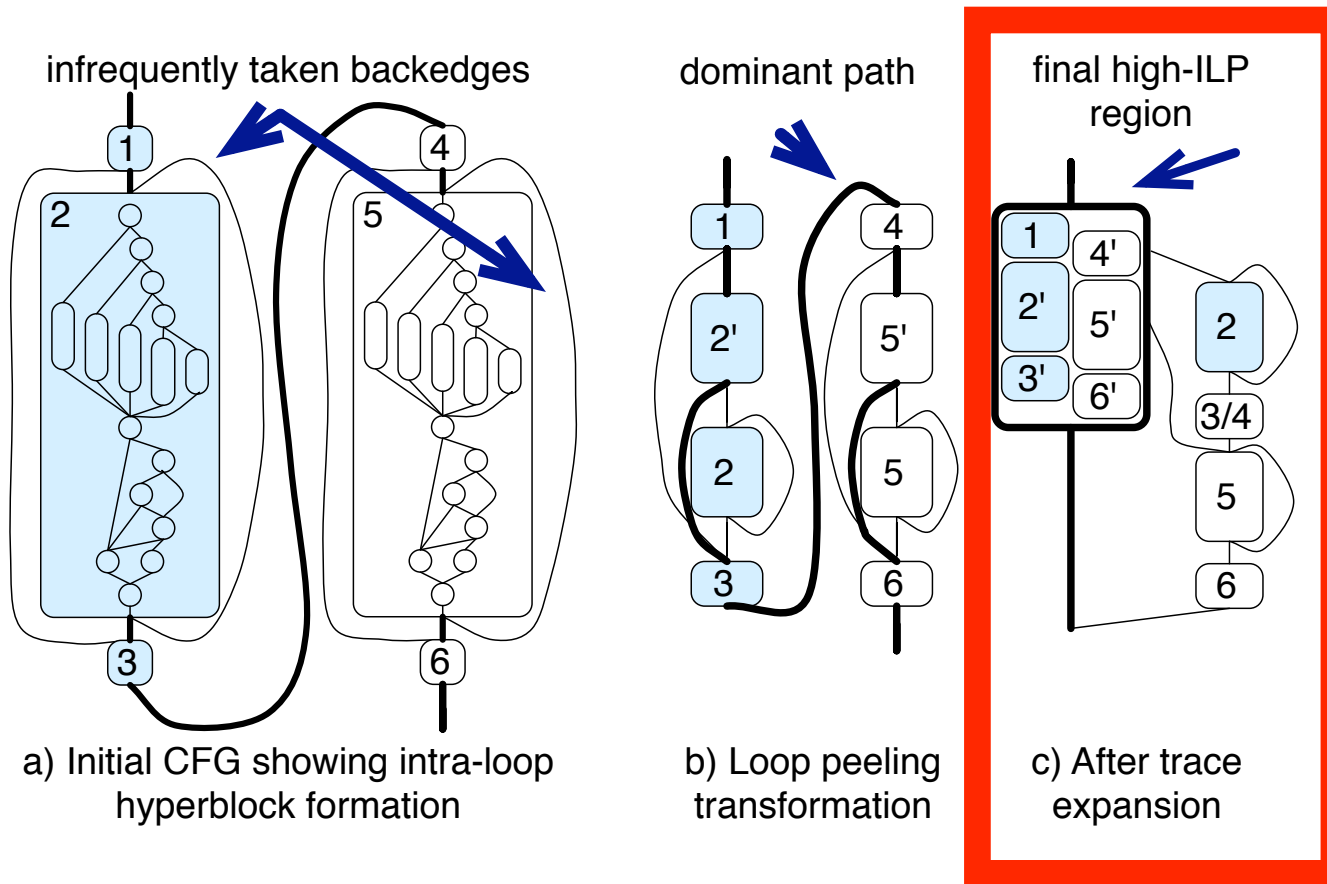
- Peeling the one expected iteration of each loop produces a straight line sequence



Example from `I86.crafty:Evaluate()` [SiasISCA04]

Combining structural transformations

- The expected path can be merged into a single hyperblock and the two loops executed in parallel

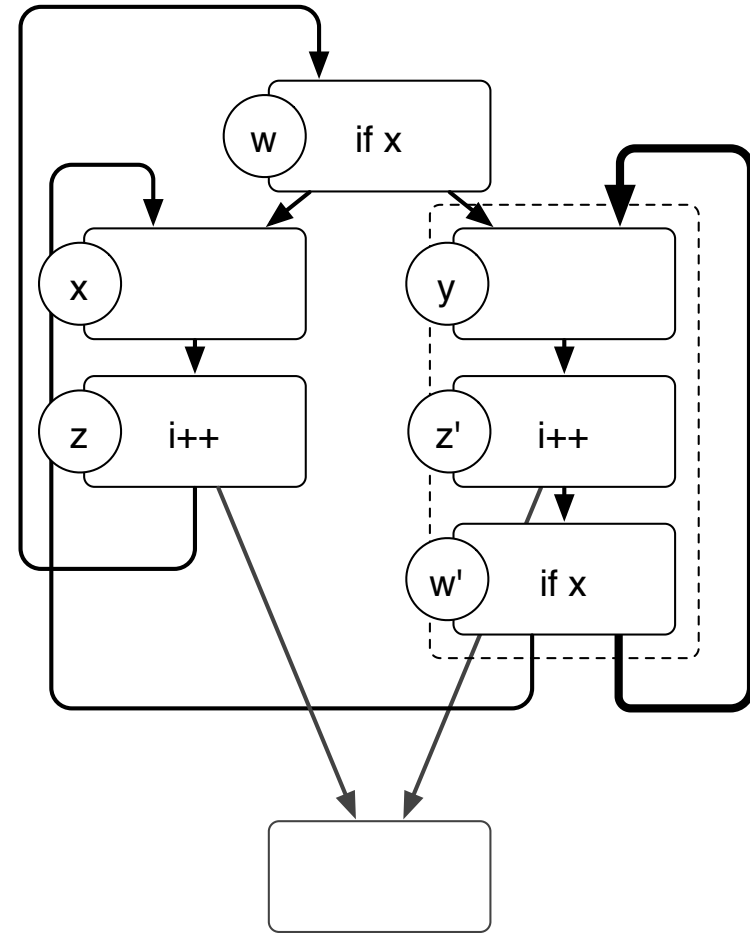
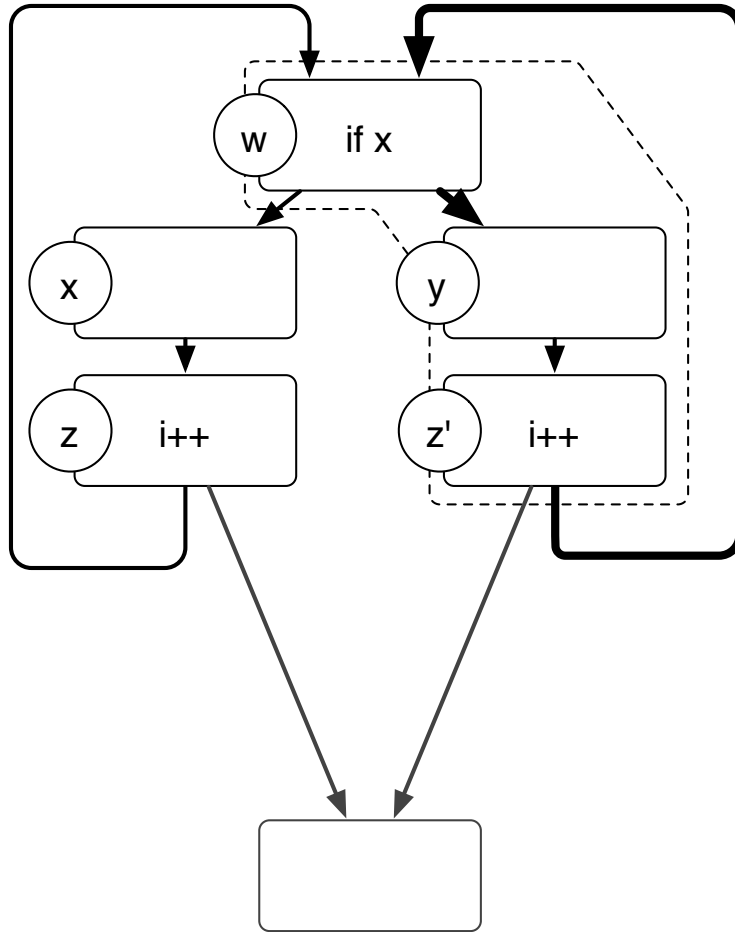


Example from `I86.crafty:Evaluate()` [SiasISCA04]

Application to GCC

- Moved Superblock formation pass to Tree-SSA level
- Perform loop header duplication during Superblock pass to form loops that can be processed by GCC's loop optimizers
- Adjust code expansion limits to allow more code duplication
- Work is available in the ia64-improvements branch in the GCC SVN repository

Loop header duplication



Structural GCC: Results

- Early results suggest early Superblock formation in GCC has a similar effect to OpenIMPACT's structural compilation model
- Case in point: 256.bzip2
 - Improves by 3% with early Superblock formation

256.bzip2

Excerpt from generateMTFValues

```
L6:  j = 0;
      tmp = yy[j];
L8:  while ( ll_i != tmp ) {
L7:      j++;
      tmp2 = tmp;
      tmp = yy[j];
      yy[j] = tmp2;
      }
L9:  yy[0] = tmp;

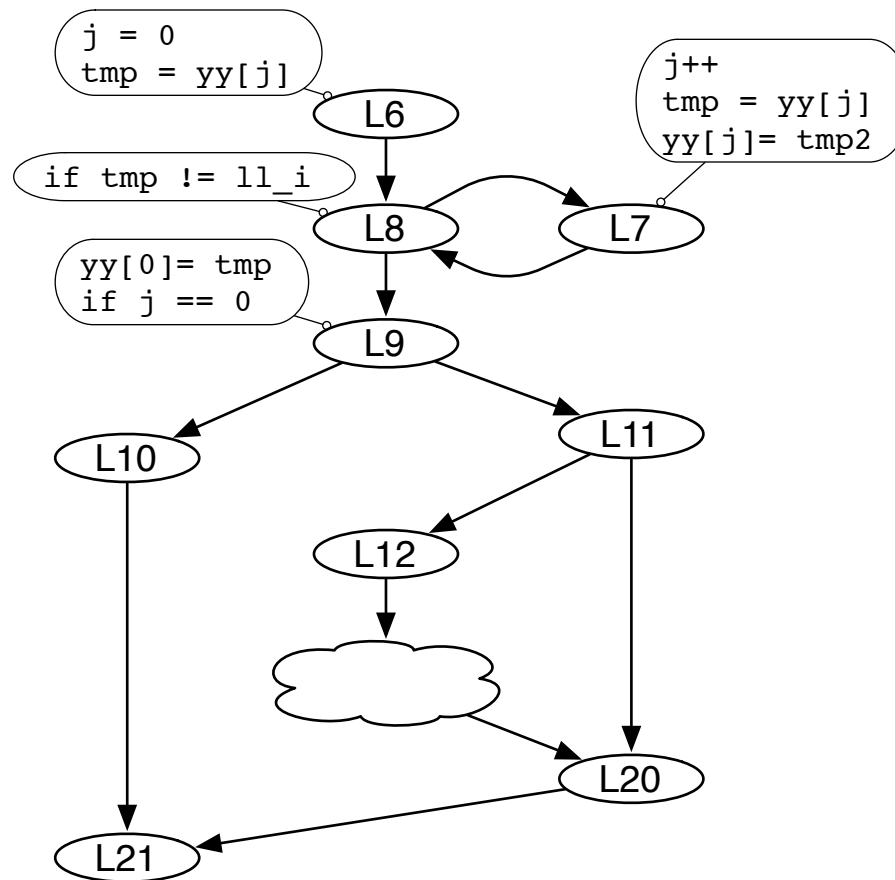
      if (j == 0)
L10: /* do something */
      /* yy is not touched */
      else
L11: /* do something else */
      /* yy is not touched */
L21: ...
```

Executes
only if loop
iterates

Closely
scheduled stores
may stall on
Itanium 2

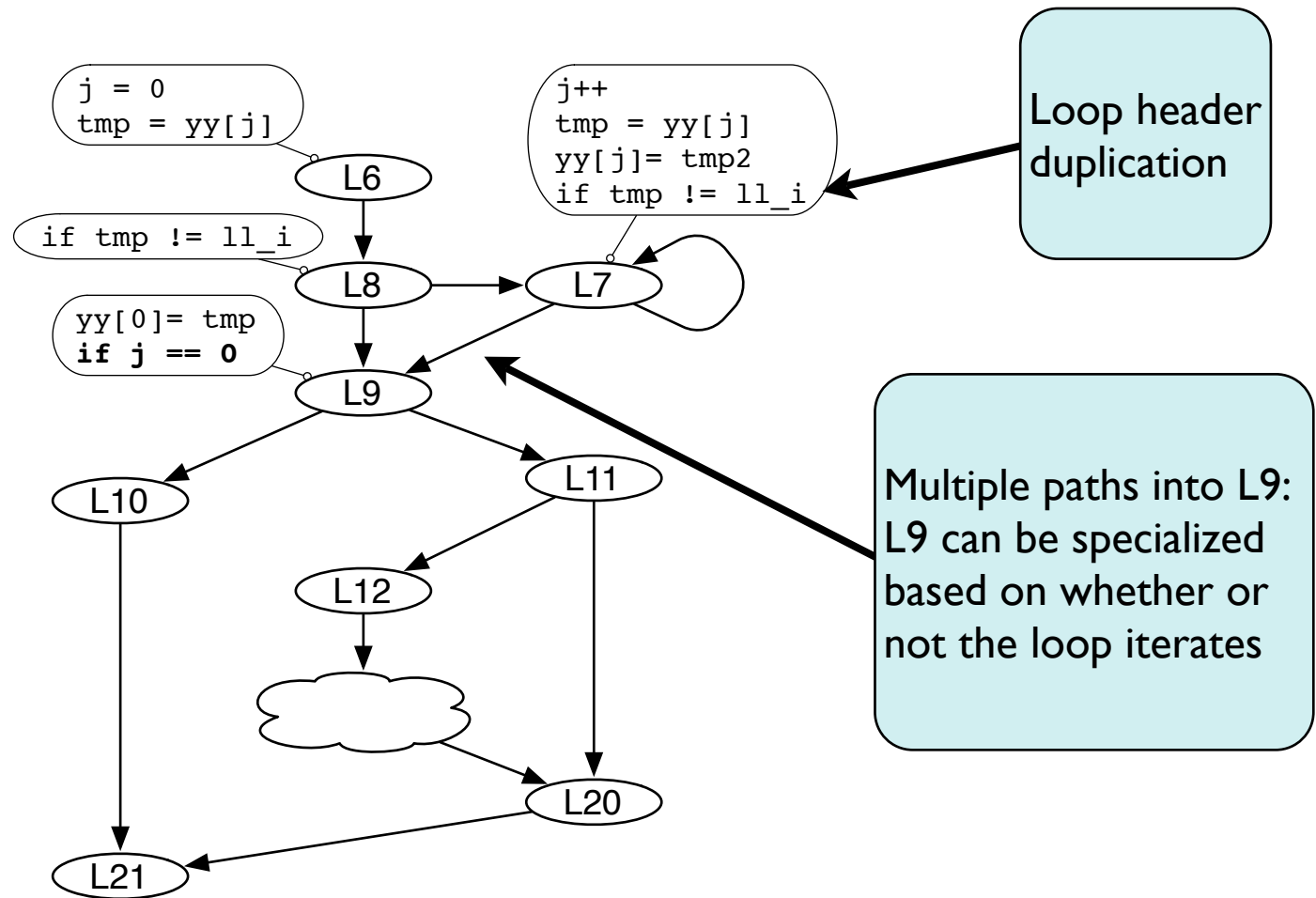
256.bzip2

Immediately after constructing SSA form



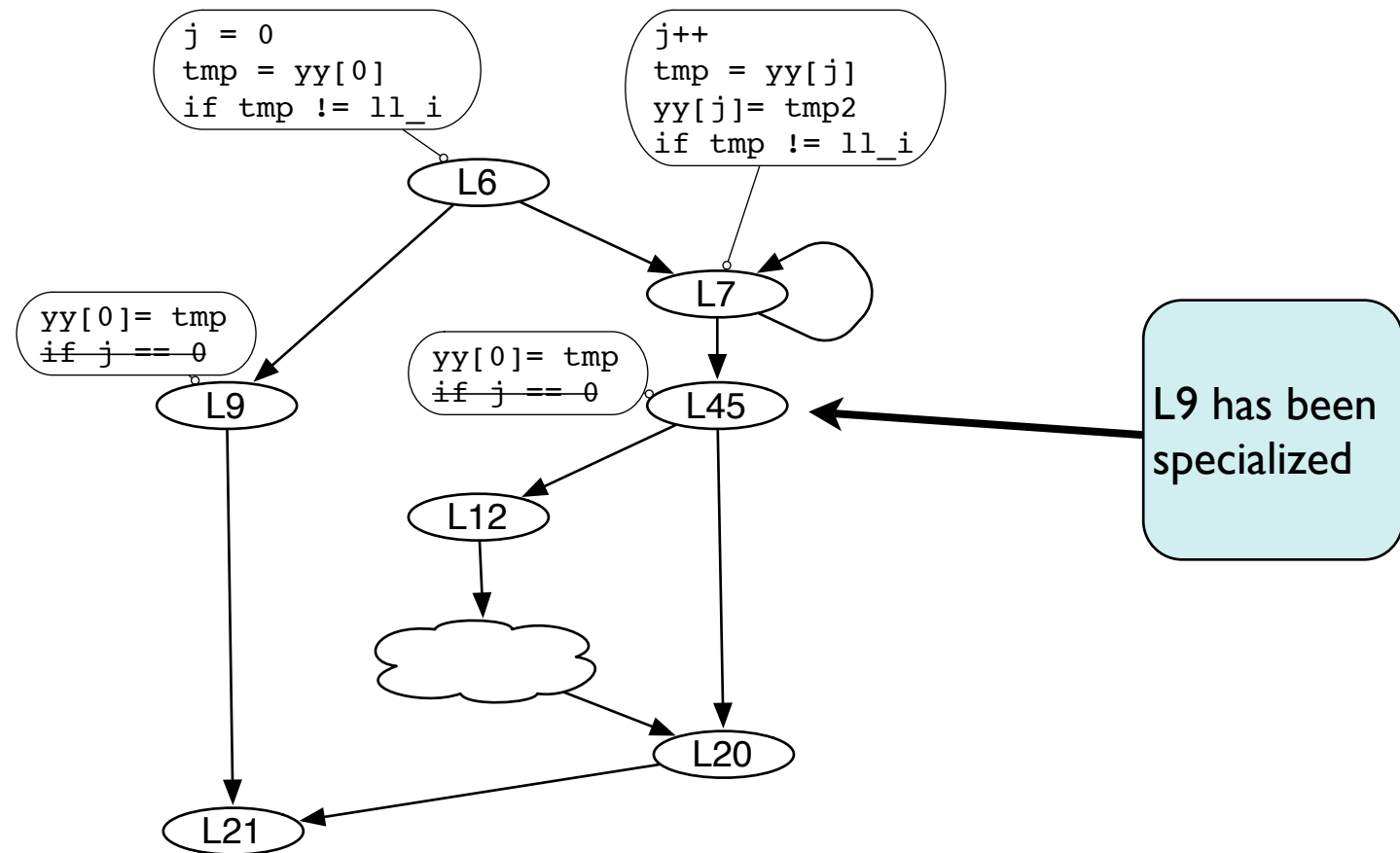
256.bzip2

After Superblock formation



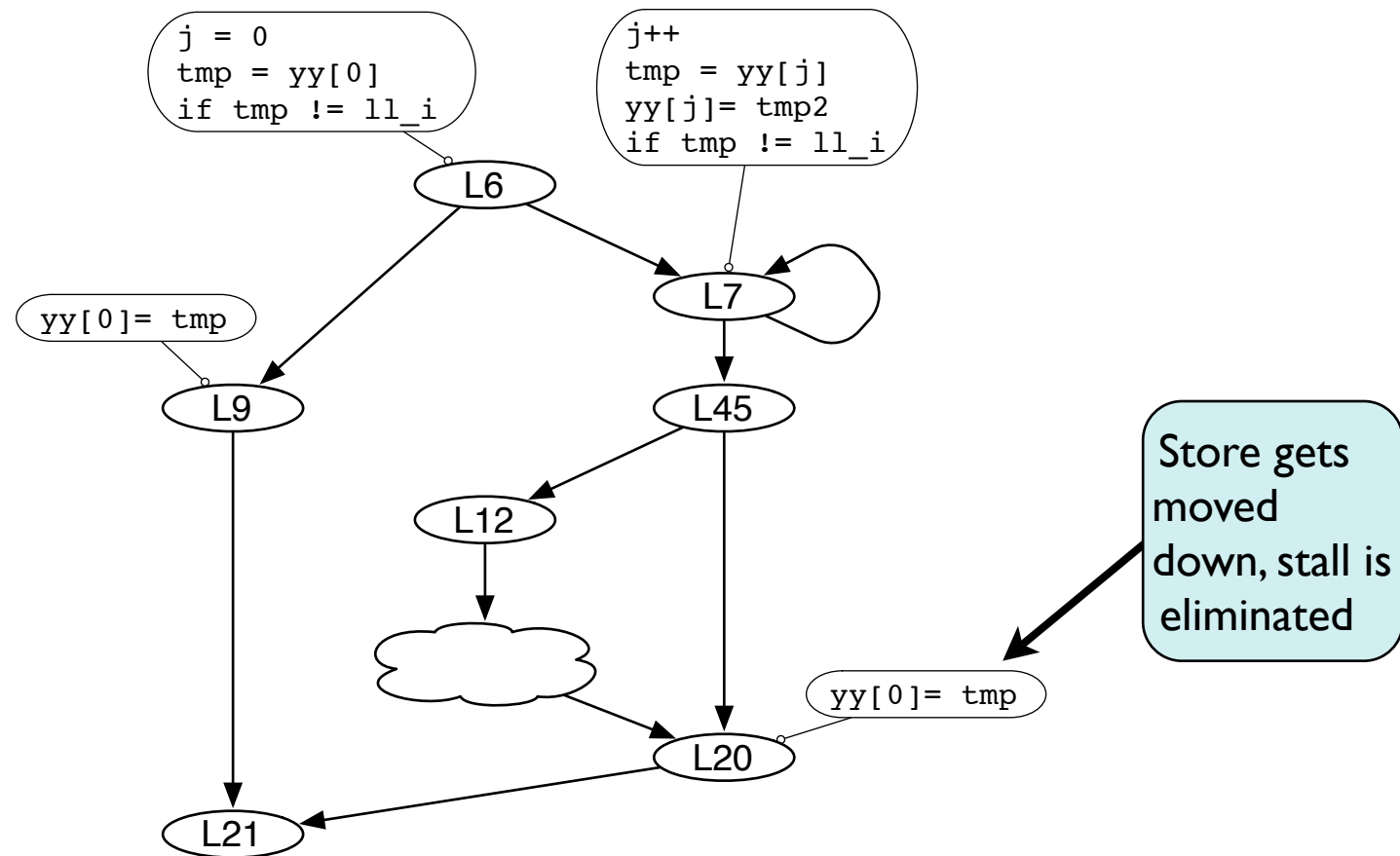
256.bzip2

After value range propagation

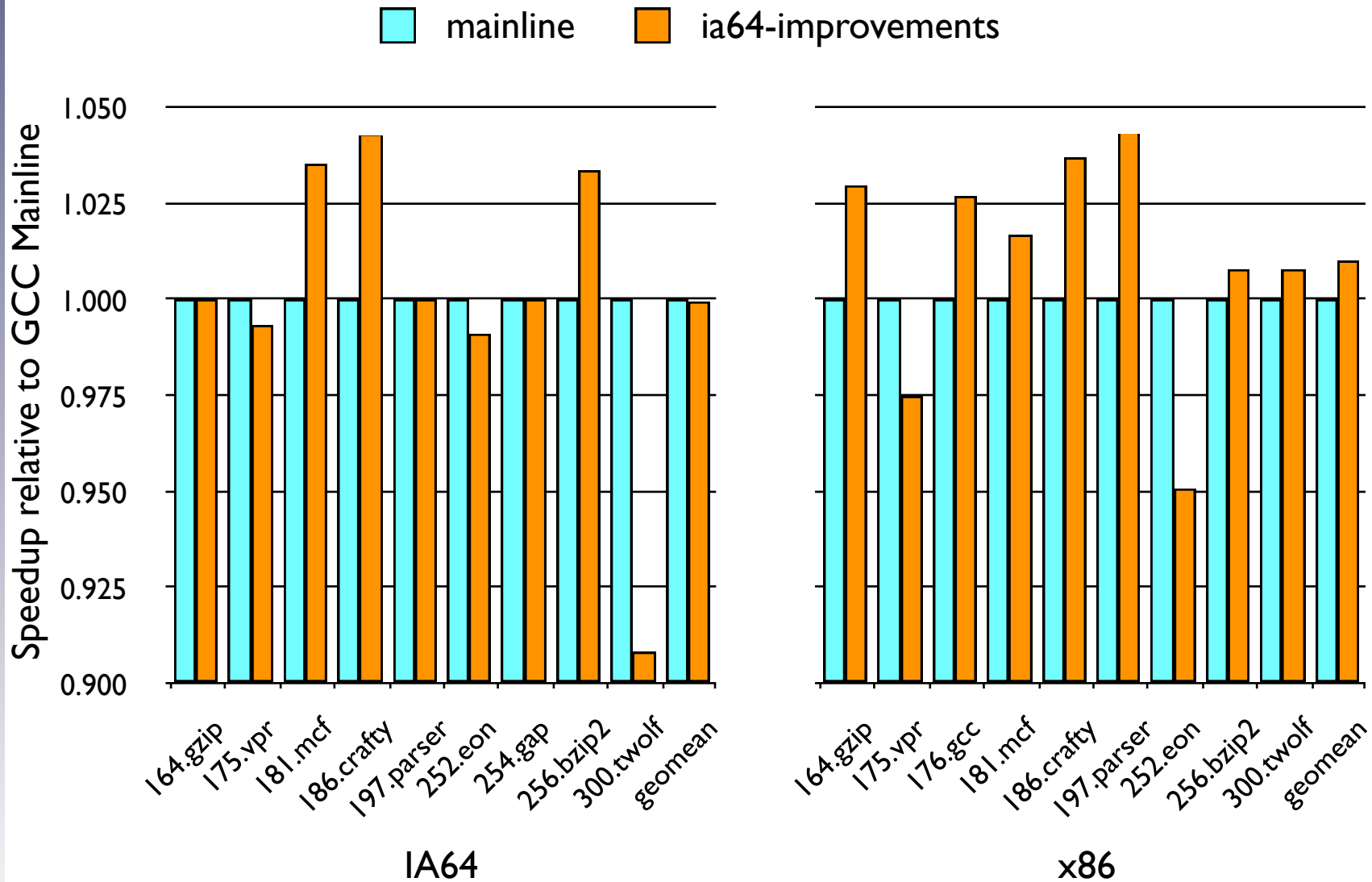


256.bzip2

After store sinking

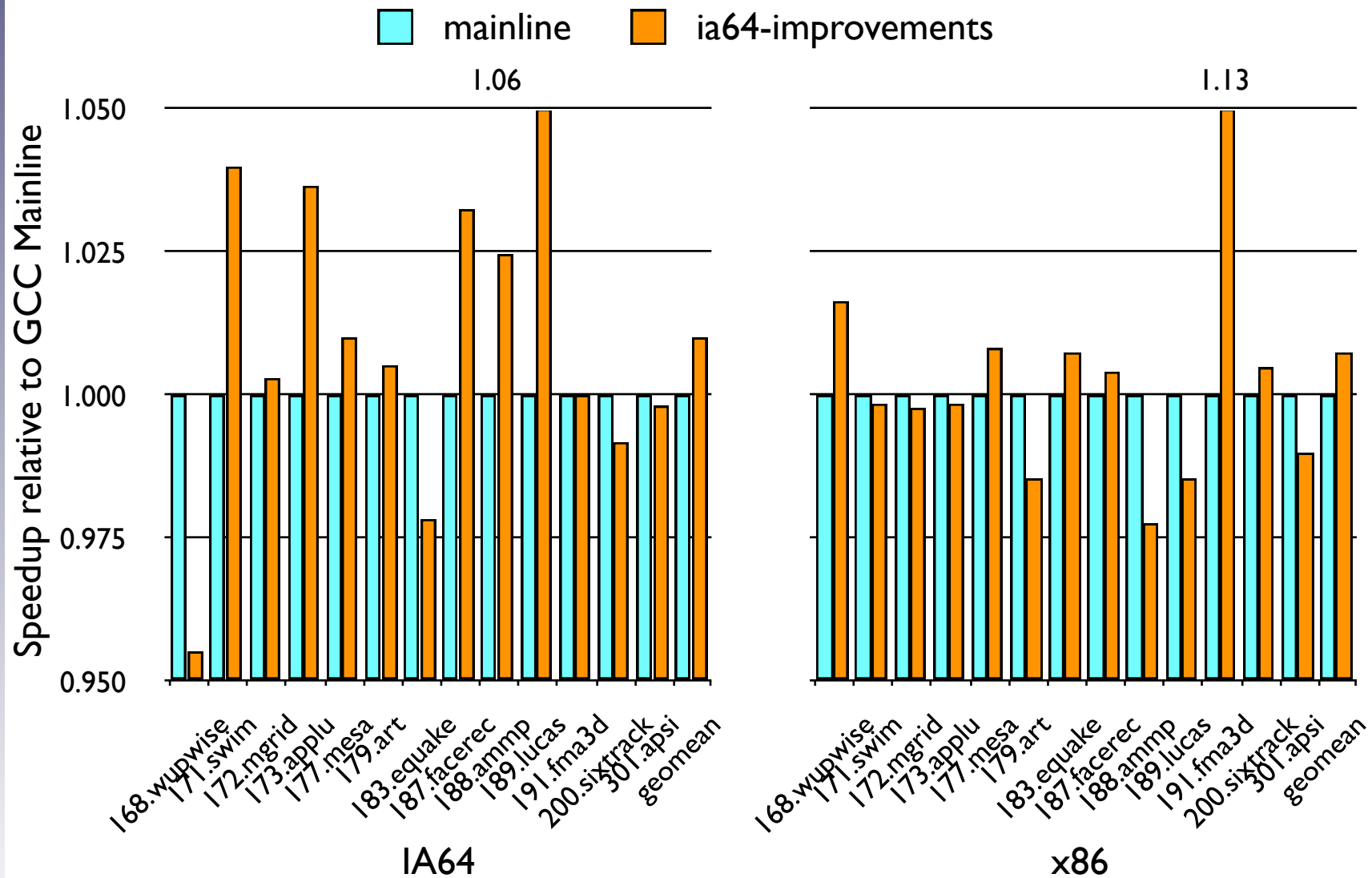


Results – Integer



Estimated SPECint2000 speedup over GCC mainline

Results – Floating Point



Estimated SPECfp2000 speedup over GCC mainline

Analysis

- 300.twolf on IA64
 - Performance degradation due to a bug in tail duplication
 - Without Superblock formation, partial redundancy elimination is able to register promote a load/store pair inside a hot loop
 - The loop in question is transformed into a non-simple form by tail duplication
- 191.fma3d on x86
 - `platq_stress_integration` sets up symbolic variables using `min/max` macros
 - Superblock formation allows constant propagation to simplify the function

Analysis: 186.crafty on IA64

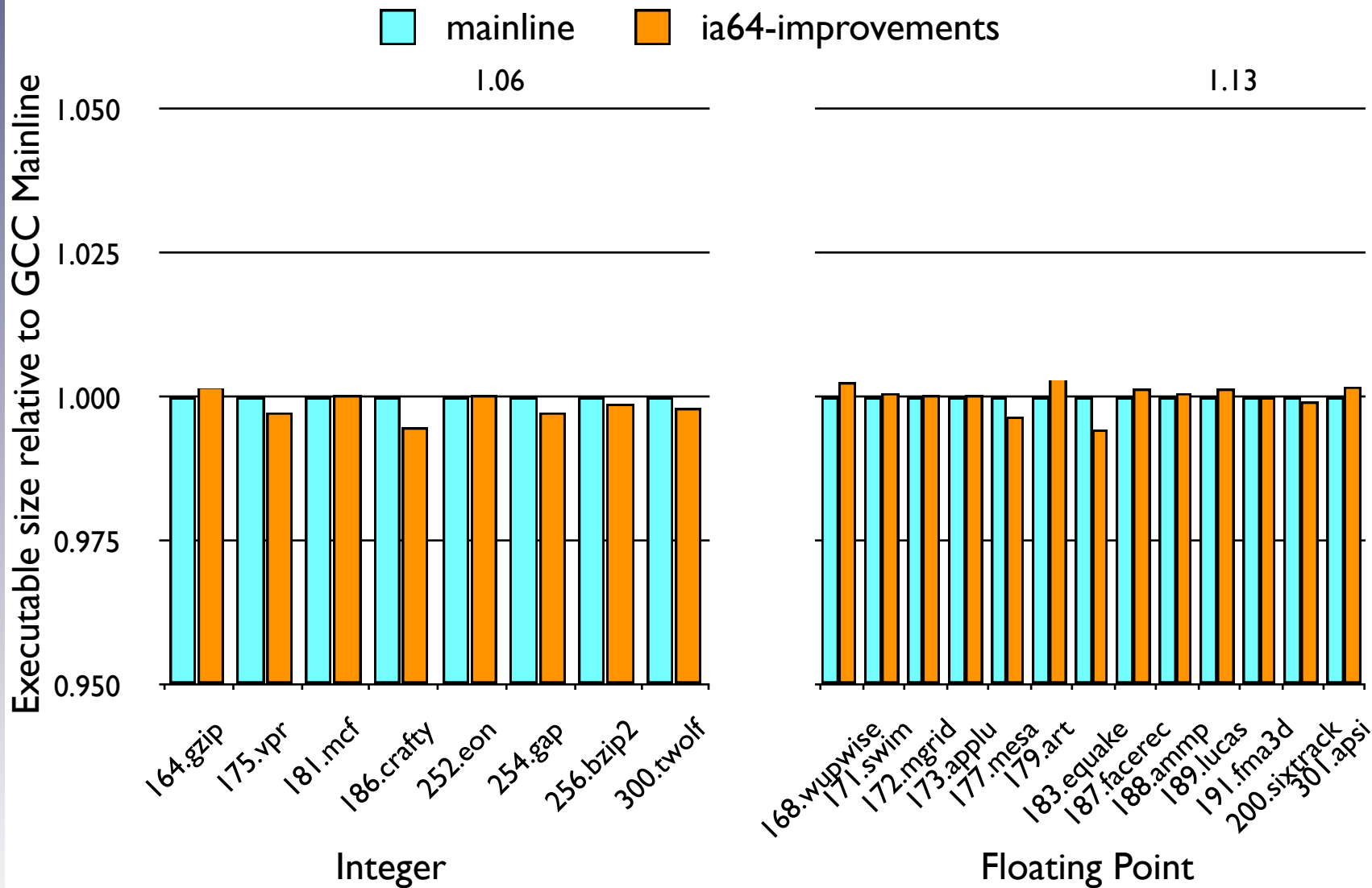
- 186.crafty improves by 4.3% when Superblocks are formed early

	Mainline	ia64-improvements
Useful instructions per cycle	1.161	1.176
Total stalls (fraction of total)	0.495	0.467
Branch mispredictions	19.3e9	18.3e9
L1I hit rate	0.808	0.809
L2I hit rate	0.997	0.998

Analysis: 186.crafty on x86

- 186.crafty improves by 3.7%
 - Instruction cache miss rate drops from 7.4 to 6.8%
 - Number of I-cache references drops by one third
 - Binary size decreases by a few KB

Results – Executable Size



IA64 executable size relative to GCC mainline

Future work

- Loop header duplication doesn't happen in some cases
- Implement other structural optimizations
- Phase ordering
- Regression testing

Conclusion

- Structural compilation is a useful technique to extract performance on both EPIC and superscalar architectures
- Aggressive code expansion in anticipation of later optimization
- Optimization specializes code, compacts the schedule
- Techniques can be ported to GCC without radically changing the compiler

Questions?

