Region-Based Compilation: An Introduction and Motivation

Richard E. Hank Wen-mei W. Hwu
University of Illinois

B. Ramakrishna Rau
Hewlett-Packard Laboratories

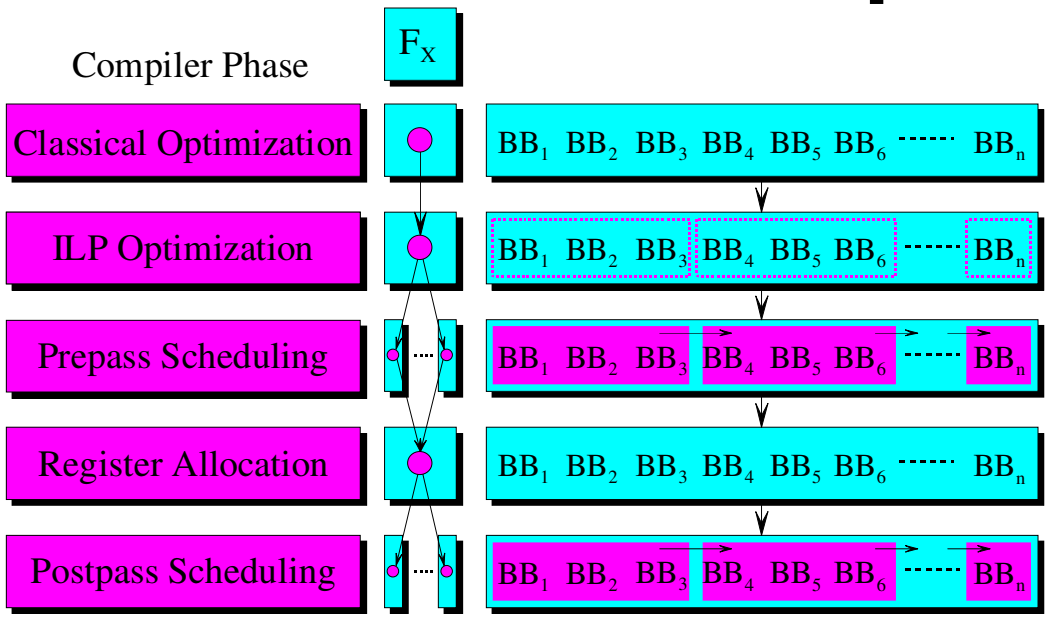
Introduction

- High-issue rate processors place great demands upon the compiler to expose ILP.
  - Requires aggressive analysis
  - Requires inlining to support inter-procedural optimization
  - Requires expensive ILP transformations
- Compiler design tradeoff
  - Aggressive transformation vs. compilation time
  - Optimizations are scaled back to reduce compilation cost
  - Production quality implementation may not reflect the true potential of a technique
- Region-based compilation reduces compilation time in the presence of aggressive transformation

Outline

- Current function-based compilation
- Benefits and problems with function inline expansion
- Region-based compilation
- Implementation status and performance results

Current Backend Compilation

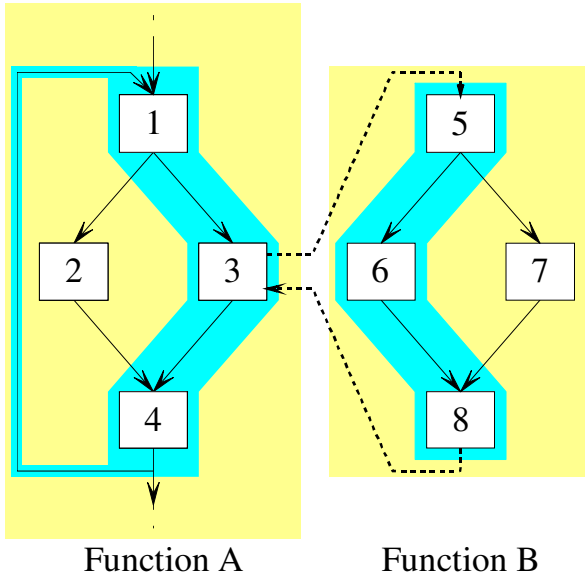


- Compilation process is built assuming functions as the unit of compilation
- Each compilation phase is typically applied to the function as a whole
- Functions are too large of a unit for scheduling
  - loops, traces, superblocks, or other regions

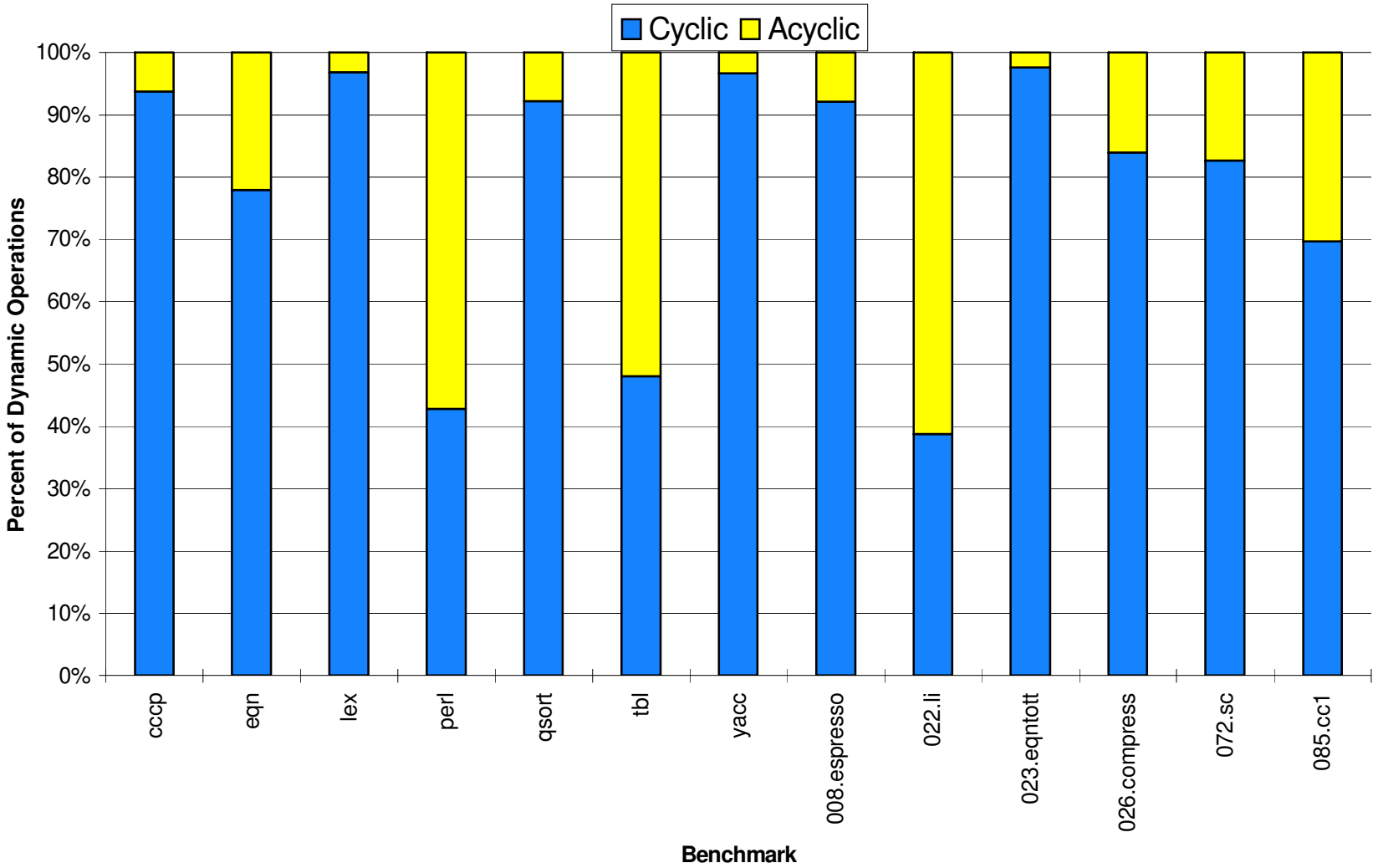
Desirable Compilation Units

- Functions are too small for analysis and many optimizations
- Functions boundaries hide information
  - Aliasing information
  - Valuable optimization opportunities
  - Control flow structure

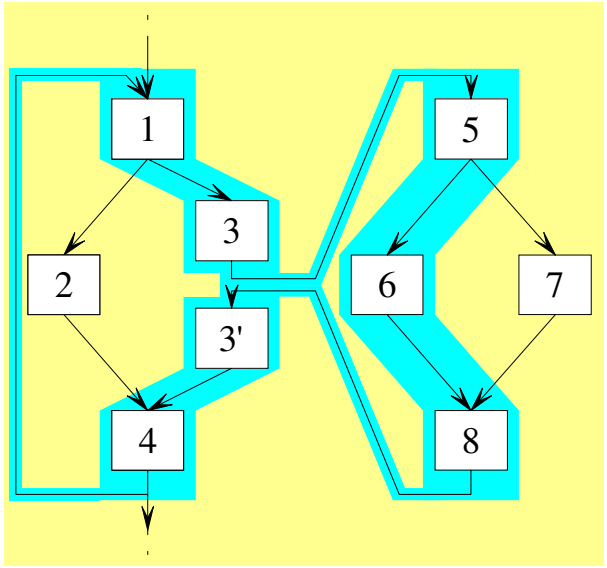
    (e.g. presence of cycles)



Intra-procedural cyclic and acyclic code distribution



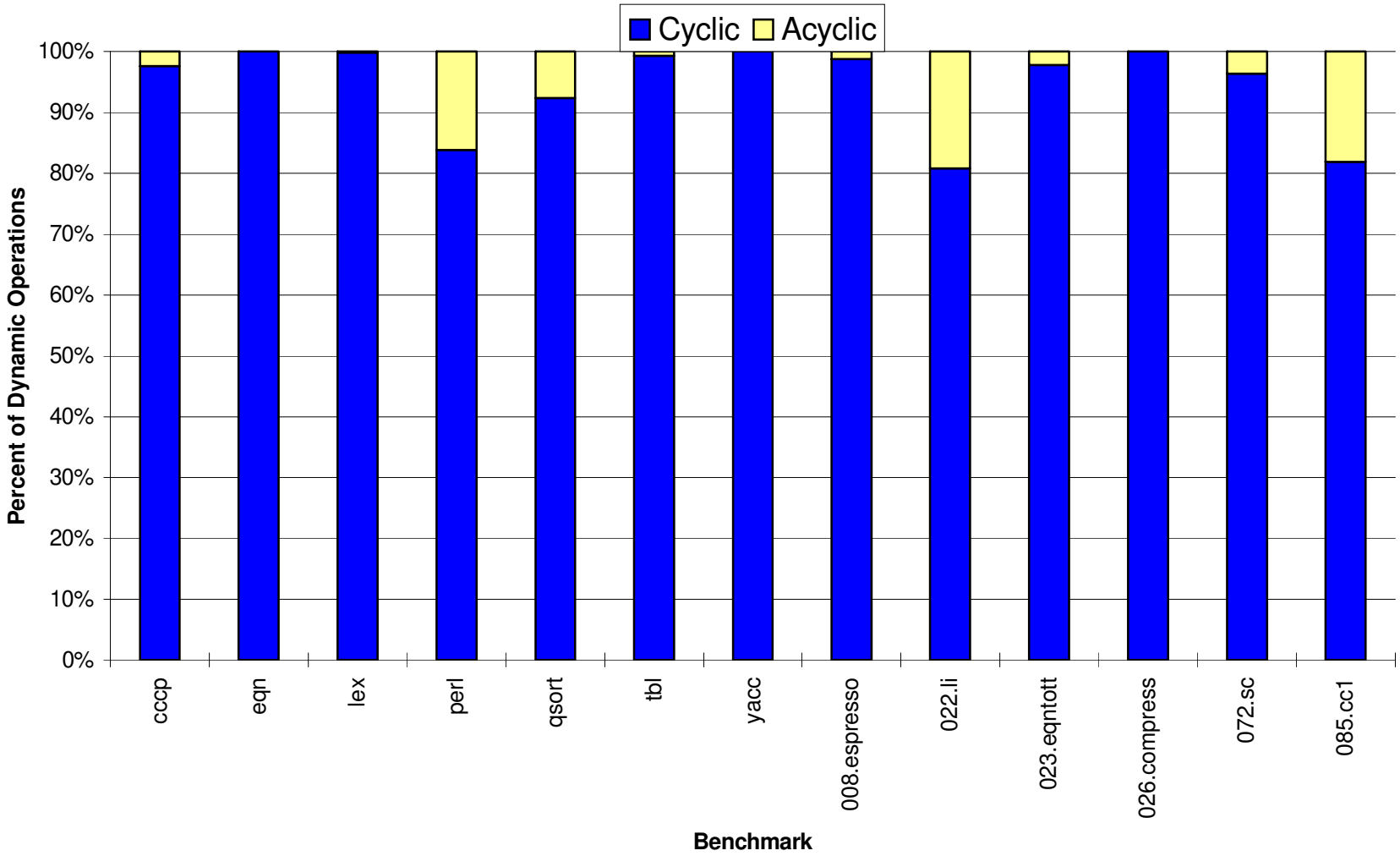
Desirable Compilation Units



Function A

- Use inlining to build better compilation units by assembling dynamically coupled functions into the same function body

Distribution of acyclic code decreases after inlining

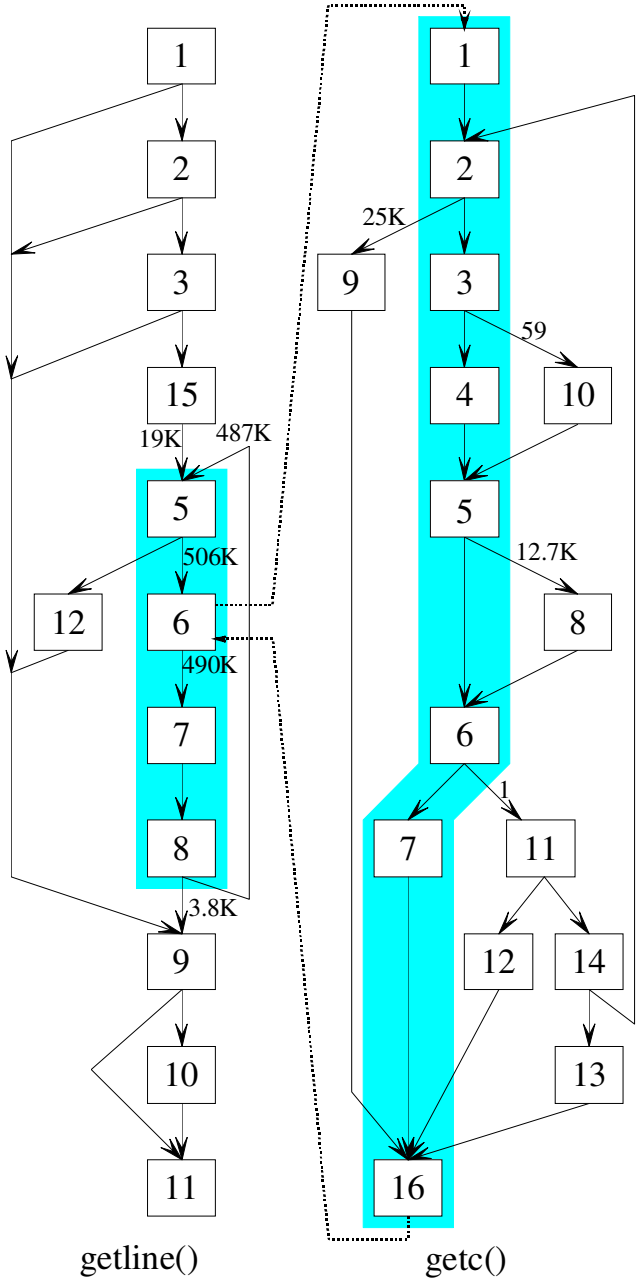


Function Inlining Benefits

- Traditionally
  - Eliminate the overhead of frequent subroutine calls
- Within an ILP compiler
  - Expose additional opportunities for application of classical optimizations
  - Improve aliasing information and subsume some inter-procedural analysis
  - Increase the amount of cyclic code visible to the compiler
    - Improved optimization and scheduling of loops

Inlining Example (eqn)

- eqn provides an example of the potential ILP benefits of inlining
- Large amount of acyclic code ~22%
- Inlining reduces this to ~0%



IMPACT

ILP Benefits of Inlining (eqn)

getline()

cb 5:	bge	r6,r11,cb12	0
	mov	r4,r6	0
	add	r6,r6,1	0
	jsr	gtc()	0
	mov	r1,$P15	1
	lsl	r14,$P15,24	1
	asr	r5,r14,24	2
	st_c	[r4+0],r5	3
	beq	r5,10,cb9	3
	beq	$P15,-1,cb9	3
	ld_i	r15,[r21+0]	1
	bne	$P15,r15,cb5	3

getc()

	mov	r46,%hi(ip)	0
	add	r25,%lo(ip),r46	1
	mov	r47,%hi(curfile)	0
	add	r26,%lo(curfile),r47	1
	mov	r48,%hi(lastchar)	0
	add	r28,%lo(lastchar),r48	1
	ld_i	r3,[0+r25]	2
	mov	r49,%hi(buf)	0
	add	r50,%lo(buf),r49	1
	bgt	r3,50,cb9	4
	ld_i	r4,[0+r26]	2
	ld_i	r43,[0+r4]	4
	add	r5,-1,r43	6
	st_i	[0+r4],r5	7
	bgt	1,r43,cb27	7
	ld_i	r9,[4+r4]	4
	add	r8,1,r9	6
	st_i	[4+r4],r8	7
	ld_uc	r11,[0+r9]	6
	st_i	[0+r28],r11	8
	beq	10,r11,cb28	8
	beq	-1,r11,cb29	8
	ld_i	$P15,[0+r28]	8

getline()

cb5:	bge	r6,r11,cb12	0
	add	r6,r6,1	0
	bgt	r19,r50,cb9	0
	add	r174,r174,-1	0
	blt	r174,0,cb88	1
	ld_uc	r178,[0+r175]	0
	add	r175,r175,1	1
	beq	r178,10,cb87	2
	beq	r178,-1,cb89	2
	mov	$P15,r178	2
	mov	r1,r178	2
	st_c	[r6-1],r178	2
	bne	r178,r15,cb5	2

Loop Invariant Code Motion

Global Variable Migration

Before Inlining

getline loop: 4 cycles

getc: 9 cycles

Loop iteration: 13 cycles

Execution Time: 7.2M cycles

After Inlining

Loop iteration: 3 cycles

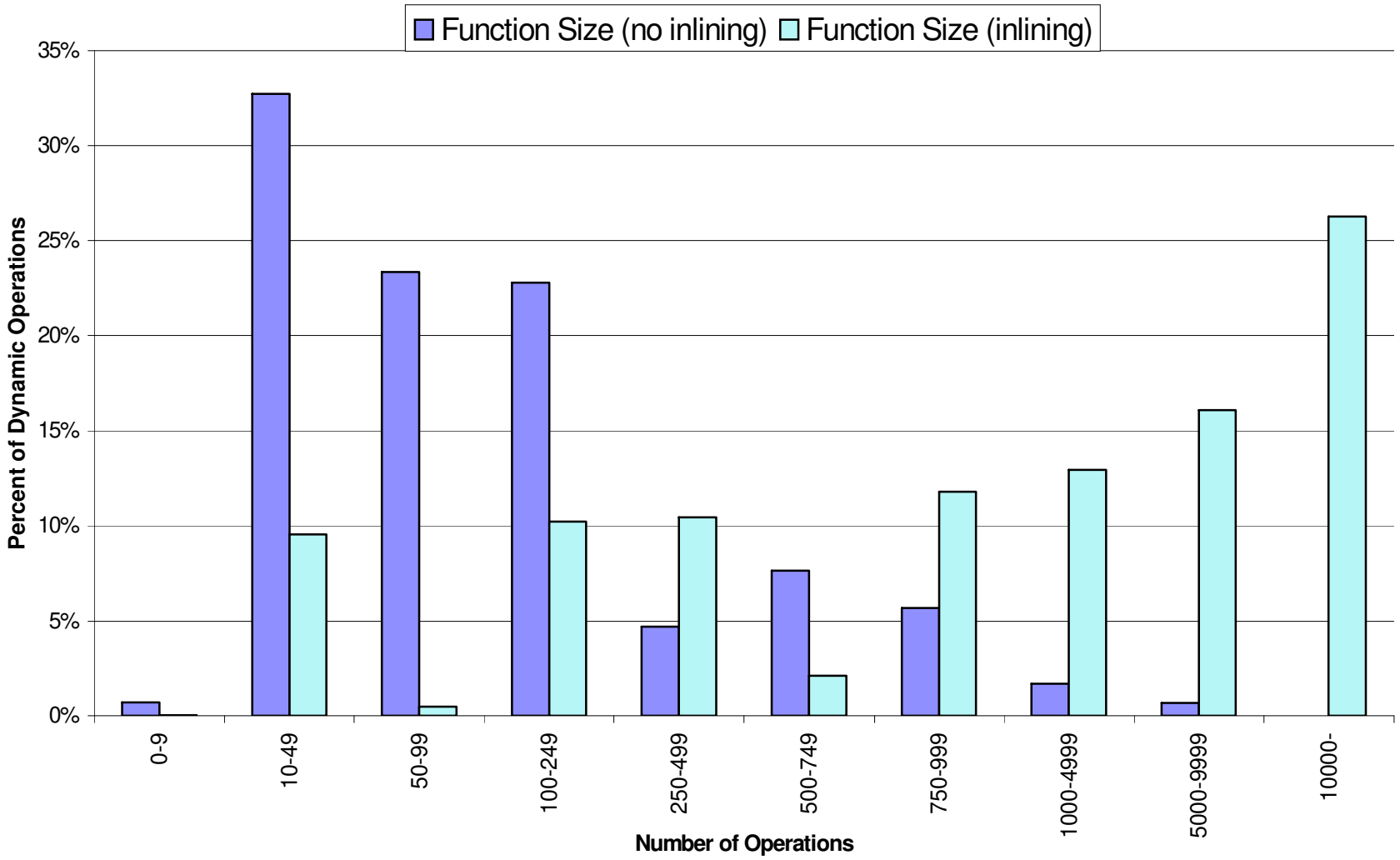
Execution Time: 1.9M cycles

IMPACT

Aggressive Inlining

- May increase register pressure to the point that the inlining benefits are negated
- May lead to excessive code expansion
  - Adversely affects compile time due to the algorithmic complexity of dataflow analysis, optimization, etc.

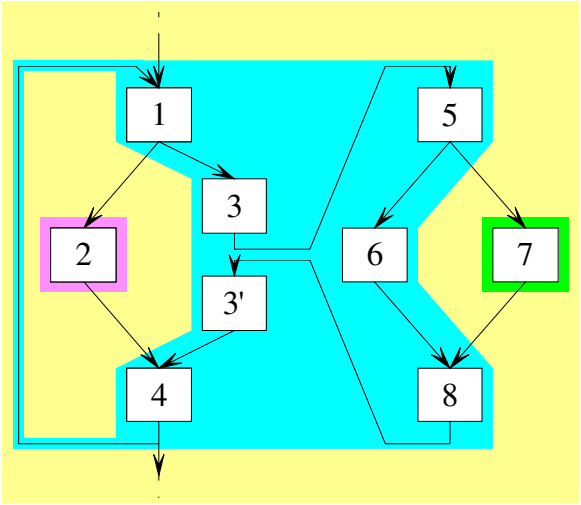
Aggressive Inlining Increases Function Size
(across all programs)



IMPACT

Region-Based Compilation Units

- Allow the compiler to repartition the program into regions.
  - Region: A collection of basic blocks and control flow arcs to be compiled as a unit.
- Compiler controls region contents
  - Representative of dynamic program behavior
  - Absence of hazards allow better optimization
    - subroutine calls, etc.
- Compiler controls region size
  - Allows the use of aggressive inlining
  - Improved compile time and memory utilization
  - Reduce importance of algorithmic complexity

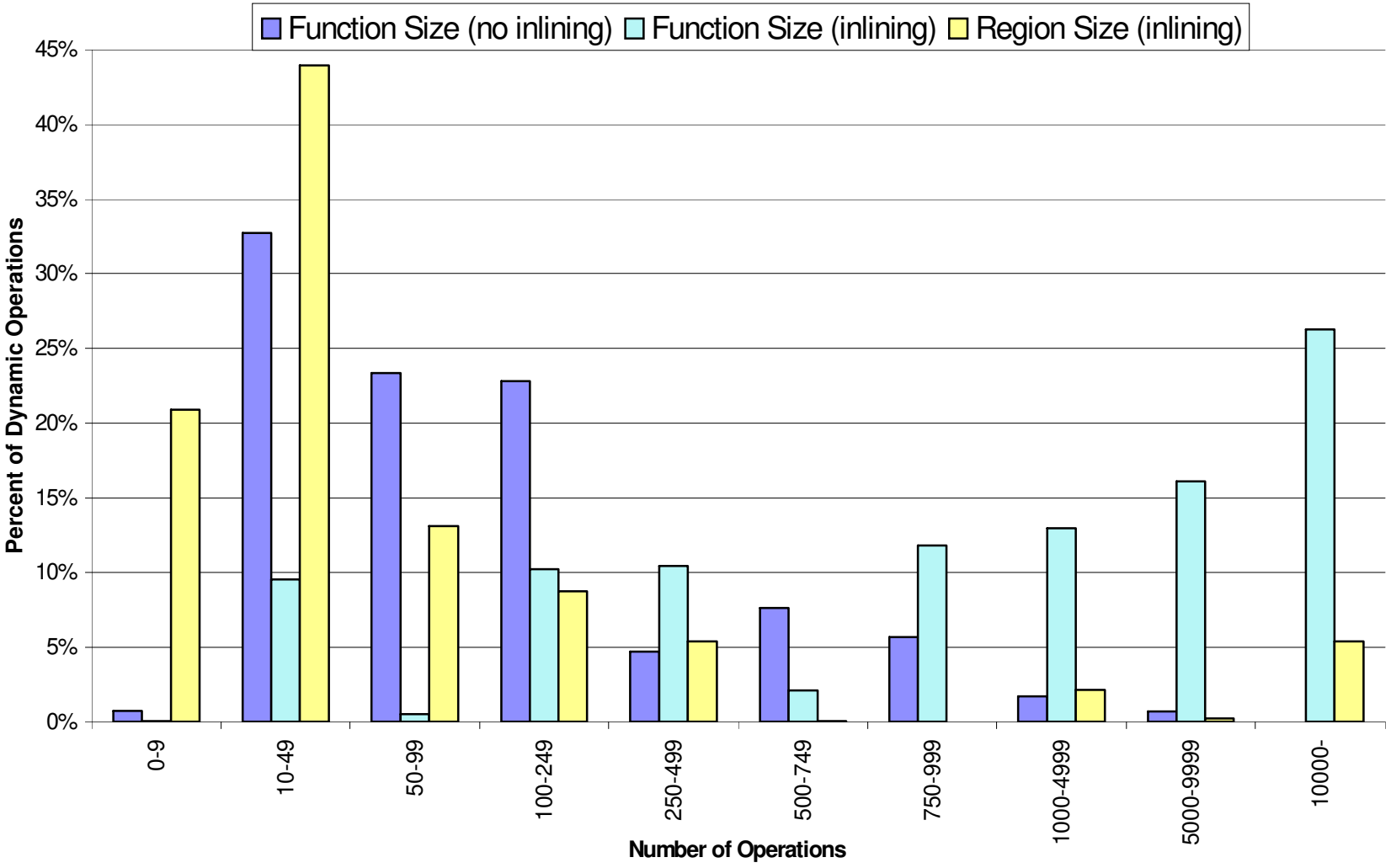


Function A

Region Formation

- Provide the best possible compilation unit for aggressive optimization
- Compiler performance
  - Consider operation count, virtual register count, number of memory dependences
- Output code quality
  - Consider presence of optimization hazards, control flow structure, dynamic program behavior
- Generalization of the profile-based trace-selection algorithm used in IMPACT

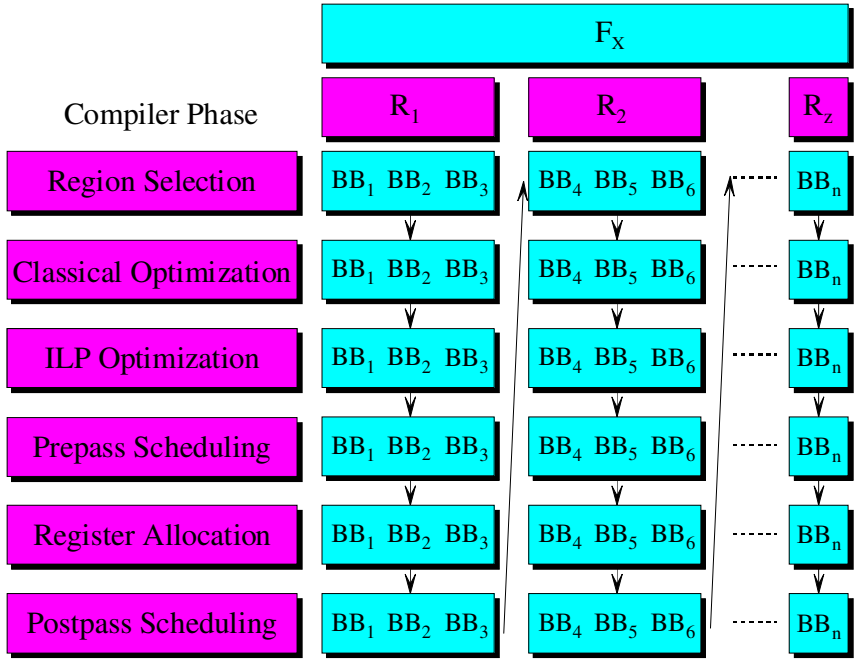
Regions Control Compilation Unit Size
(across all programs)



IMPACT

Region-Based Compilation

- Implications to compilation
  - Vertical compilation within the function body possible
    - Basic blocks in different regions may be in different phases of compilation
  - Transformations selected using region characteristics
  - Potential degradation in output code quality
  - Maintain boundary conditions



Region Boundary Conditions

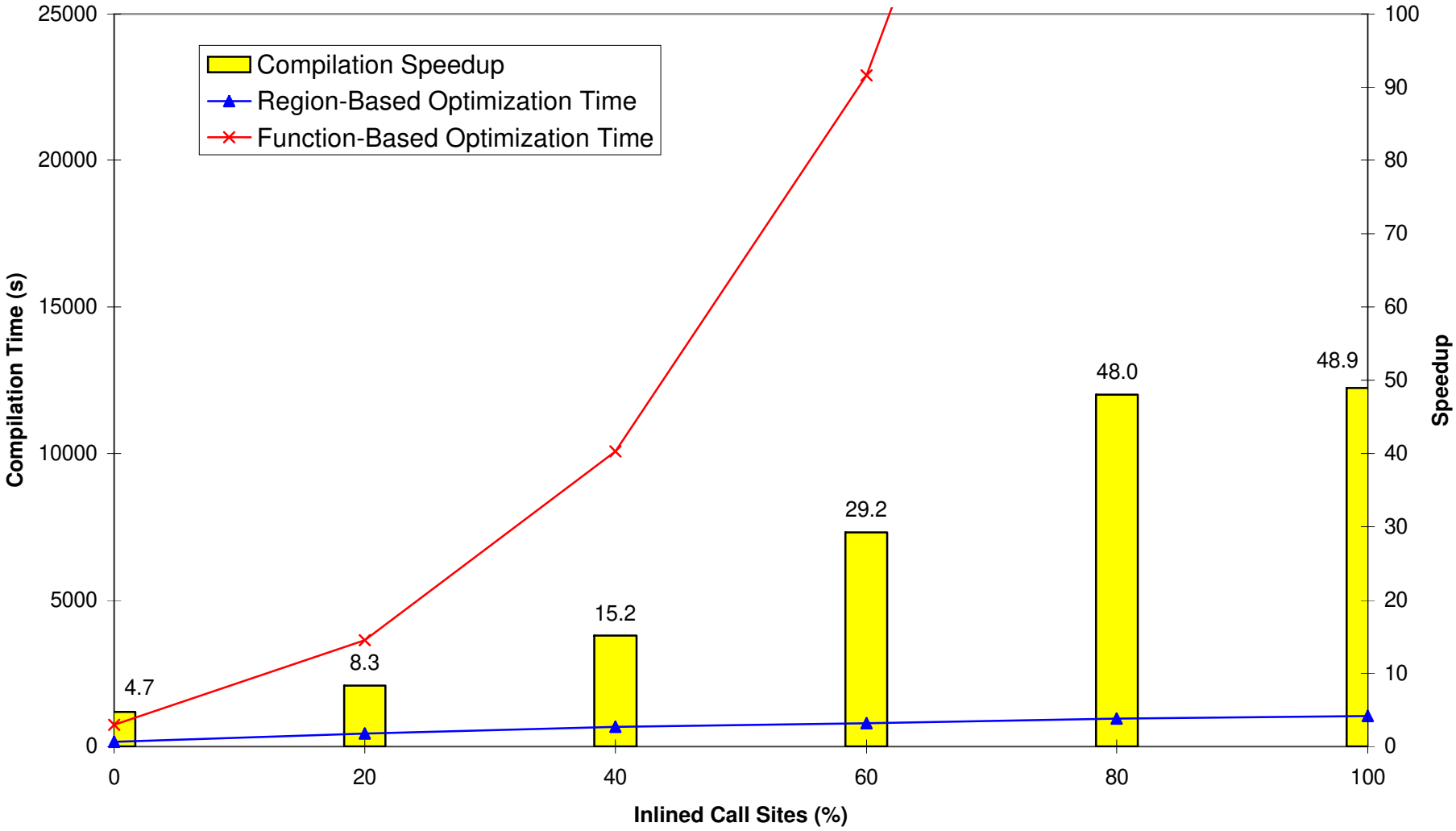
- Function boundary conditions are defined by the parameter passing convention
- Separate compilation of regions requires maintenance of
  - Variable Liveness
  - Variable Availability
  - Memory dependences
  - Scheduling constraints [Multiflow]
  - Register allocation constraints [Multiflow]

Region-Based Compilation: IMPACT

Static/Dynamic Profiling	
Function Inline Expansion	
Interprocedural Array Dependence and Pointer Analysis	
Region Formation	
Classic Local, Global, and Loop Optimizations	
Superblock Formation and Enlargement	
Superblock Optimization	
Register Renaming Accumulator Variable Expansion	Loop Unrolling Arithmetic Height Reduction
Superblock Acyclic Scheduling	Superblock Modulo Scheduling
Register Allocation	
Postpass Scheduling	

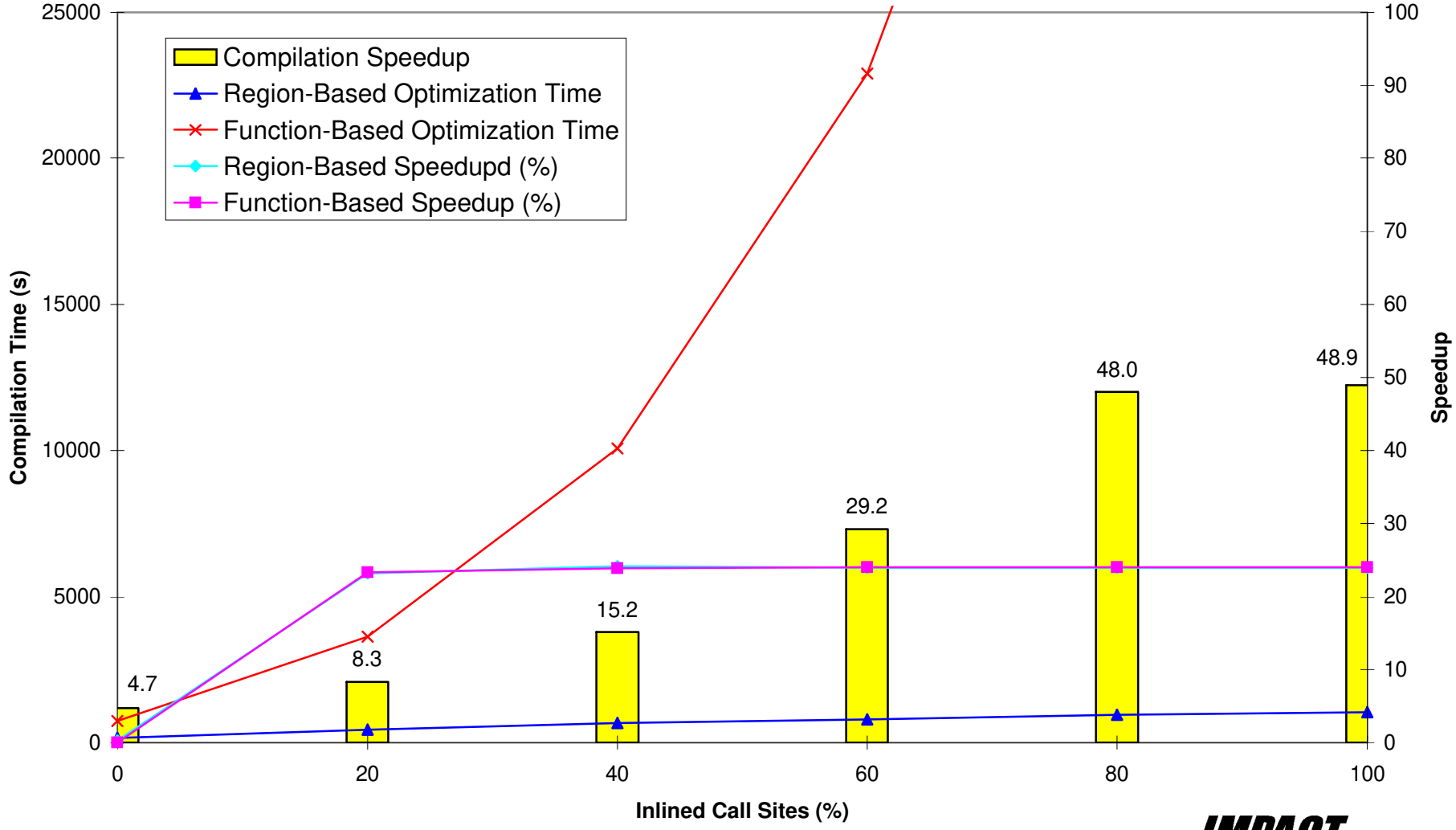
Region-Based Compilation: Reality

Comparison of Region-Based and Function-Based Compilation for perl



IMPACT

Region-Based Compilation: Reality



IMPACT

Region-Based Compilation: Benefits

- Region size is compiler controlled
  - Reduce the importance of the algorithmic complexity of employed algorithms
  - Compiler is faster and easier to design
- Reduce phase ordering effects
  - Optimization of compensation code
  - Applied transformations selected based on region characteristics
    - topology, content, previous transformations, etc.
- Profile sensitive regions
  - May allow better optimization
  - May allow simpler algorithms