WCET Code Positioning

Wankang Zhao and David Whalley
Florida State University

Christopher Healy
Furman University

Frank Mueller
North Carolina State University

Why Reduce the WCET?

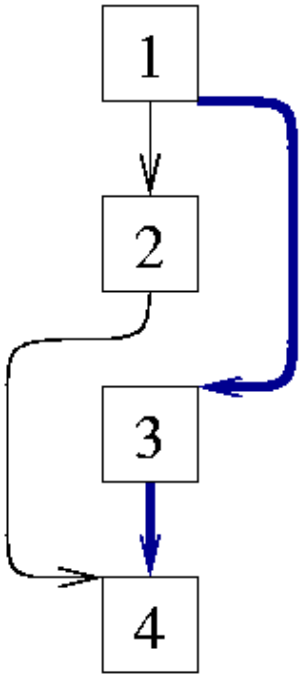
- A task is more likely to meet timing constraints.
- Can lower the clock rate to:
  - reduce the power consumption
  - while still meeting the timing constraints

Code Positioning

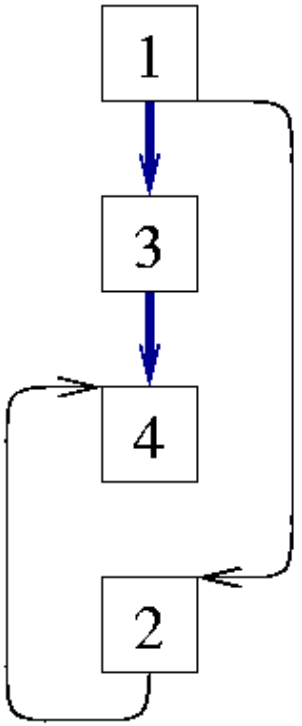
- Some machines incur a pipeline delay for every transfer of control.
  - taken branches, unconditional jumps, etc.
- A basic block consists of a sequence of instructions that has:
  - a single entry at the top
  - a single exit at the bottom
- A code positioning optimization attempts to find the best ordering of the basic blocks to minimize the number of:
  - taken branches
  - unconditional jumps

ACET Code Positioning

- Uses run-time data to calculate the frequencies of the edges (transitions) between basic blocks.
- Orders basic blocks so that the most frequently traversed edges are contiguous in memory.



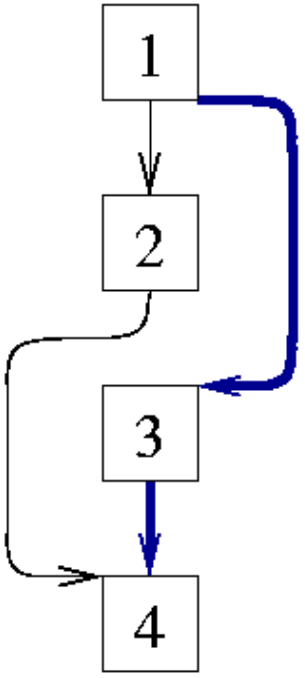
Before Positioning



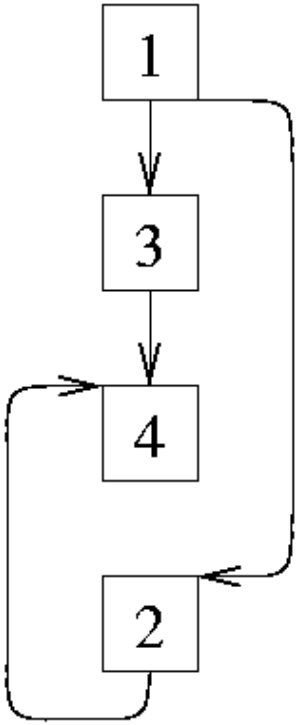
After Positioning

WCET Code Positioning

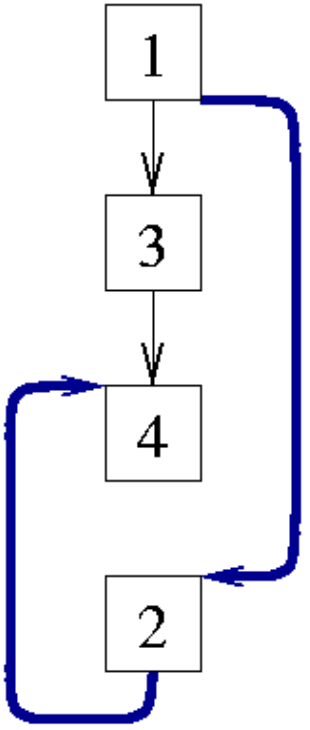
- Make edges along the WC path contiguous.
- Problem is that it may result in a new WC path.



WC Path



After Positioning



New WC Path

Outline of Rest of Presentation

- Related Work
- Research Framework
- WCET Code Positioning
- WCET Target Alignment
- Experimental Results
- Future Work
- Conclusions

Related Work on Reducing WCET

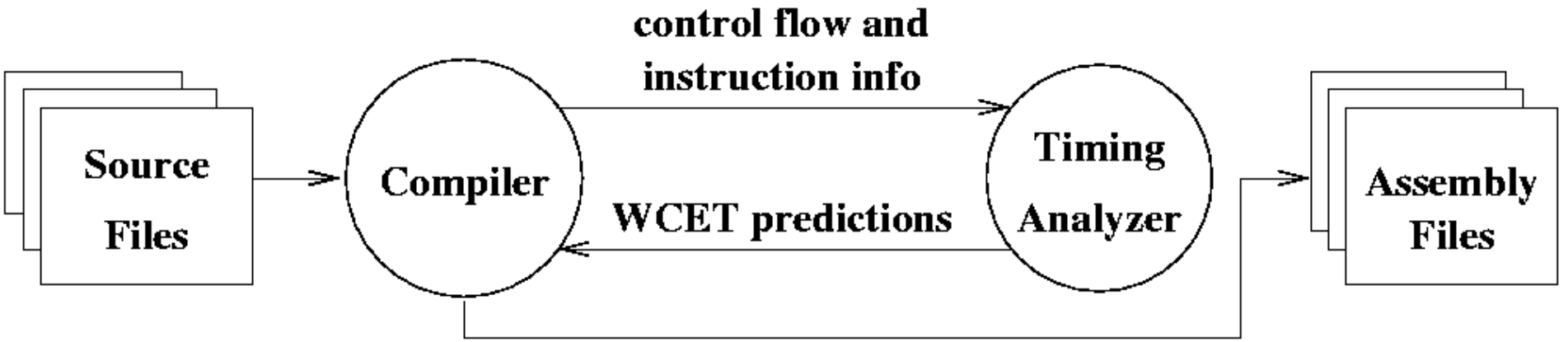
- reducing WCET in critical sections
  - Marlowe and Masticola, System Integration '92
  - Hong and Gerber, PLDI '93
- reducing WCET on a dual instruction set processor
  - Lee, et. al., SCOPES '04
- tuning WCET by searching for effective optimization sequences
  - Zhao, et. al., RTAS '04

Related Work on Code Positioning

- improving instruction cache performance
  - McFarling, ASPLOS '89
  - Hwu and Chang, ISCA '89
  - Pettis and Hansen, PLDI '90
- reducing transfer of control penalties
  - Calder and Grunwald, ASPLOS '94
  - Young, et. al., PLDI '97

Overview of the Worst-Case Compilation Process

- Timing analyzer invoked on demand by the compiler.
- Returns the WCET and the WC path information to the compiler.

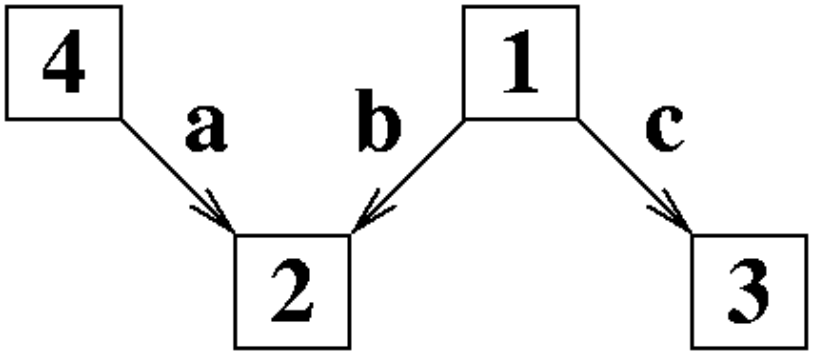


StarCore SC100 Processor

- digital signal processor
- no caches and no operating system
- simple five stage pipeline
  - one to three cycle penalty on each transfer of control
  - one cycle penalty for misaligned targets

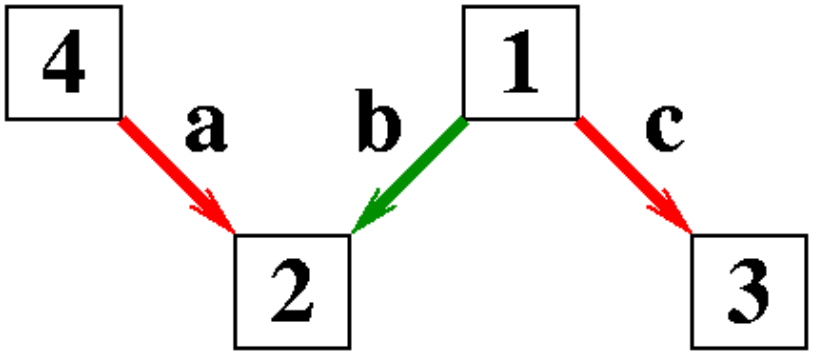
Contiguous and Noncontiguous Edges

- A directed edge connecting two basic blocks is contiguous if the source block is immediately followed by the target block in memory.
- Not all edges can be contiguous.



Contiguous and Noncontiguous Edges

- A directed edge connecting two basic blocks is contiguous if the source block is immediately followed by the target block in memory.
- Not all edges can be contiguous.



Definitions

- unpositioned edge
  - an edge not yet classified as contiguous or noncontiguous
- UB-WCET (upper bound WCET) of a path
  - All current unpositioned edges are assumed to be noncontiguous.
- LB-WCET (lower bound WCET) of a path
  - All current unpositioned edges are assumed to be contiguous.
- noncontributing path
  - The UB-WCET is less than the LB-WCET of another path at the same loop or function level.

WCET Code Positioning Algorithm

(1) Mark all edges as unpositioned.

(2) Calculate UB-WCET and LB-WCET for each path.

(3) Sort paths by if it is contributing, its loop nesting, its UB-WCET, and its LB-WCET.

(4) Select first path in list with an unpositioned edge.

(5) Select an edge to make contiguous that minimizes the UB-WCET of this path. Break ties by selecting an edge that lowers the UB-WCET of the next encountered path. Break remaining ties by selecting an edge that results in the smallest LB-WCET increase of the next encountered path.

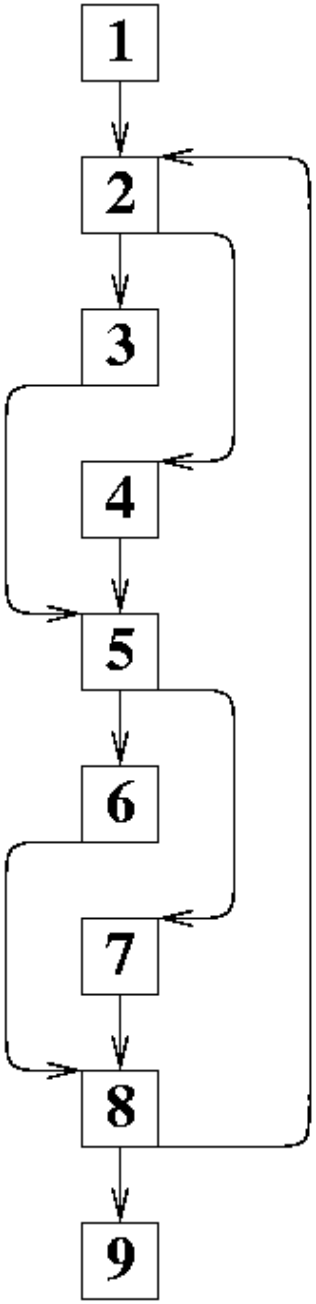
(6) Mark the selected edge as contiguous. Mark other edges that become noncontigous.

(7) Go to step 2 if any remaining unpositioned edges.

WCET Code Positioning Example

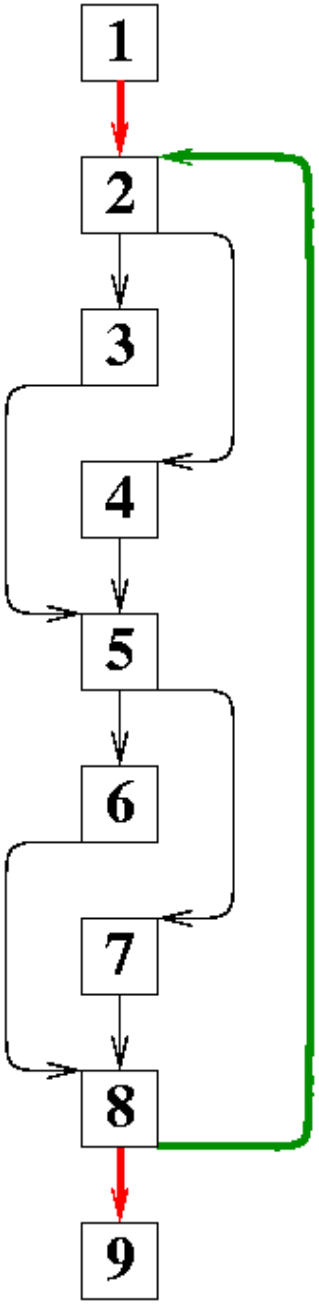
```
...
for (i=0; i< 1000; i++) {
   if (a[i] < 0)
      a[i] = 0;
   else {
      a[i] += 1;
      sumalla += a[i];
   }
   if (b[i] < 0)
      b[i] = 0;
   else {
      b[i] += 1;
      sumallb += b[i];
      b[i] = a[i] - 1;
   }
}
...
```

Example of Positioning Edges



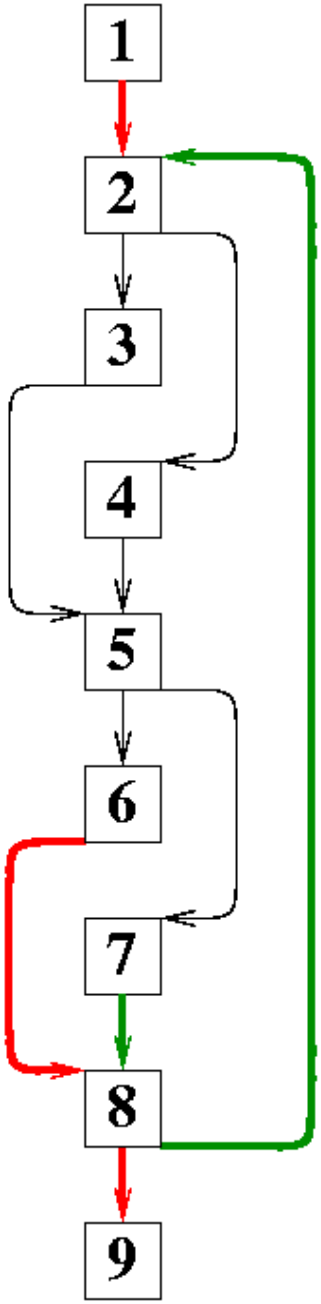
	UB–WCET	LB–WCET	
Path A:	36	21	2 → 3 → 5 → 6 → 8 →
Path B:	40	25	2 → 3 → 5 → 7 → 8 →
Path C:	37	22	2 → 4 → 5 → 6 → 8 →
Path D:	41	26	2 → 4 → 5 → 7 → 8 →
Path E:	37,020	22,018	1 → loop → 9

Example of Positioning Edges (Step 1)



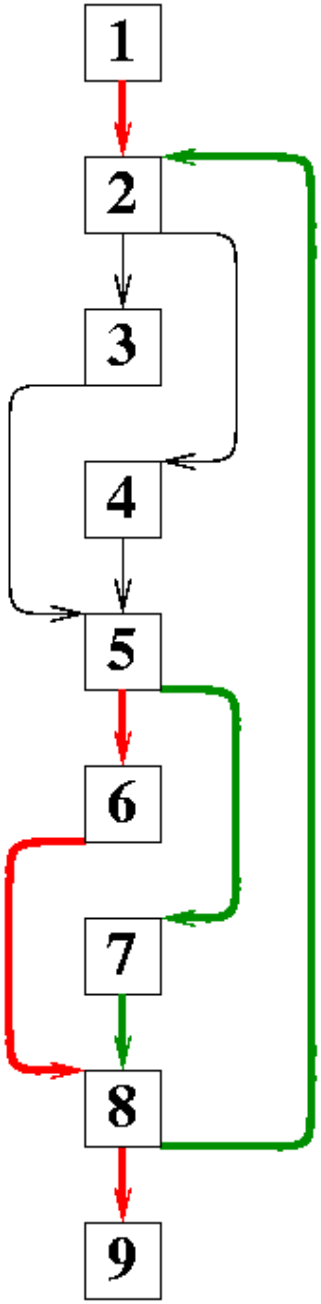
	UB–WCET	LB–WCET	
Path A:	33	21	2 → 3 → 5 → 6 → 8 →
Path B:	37	25	2 → 3 → 5 → 7 → 8 →
Path C:	34	22	2 → 4 → 5 → 6 → 8 →
Path D:	38	26	2 → 4 → 5 → 7 → 8 →
Path E:	34,024	22,024	1 → loop → 9

Example of Positioning Edges (Step 2)



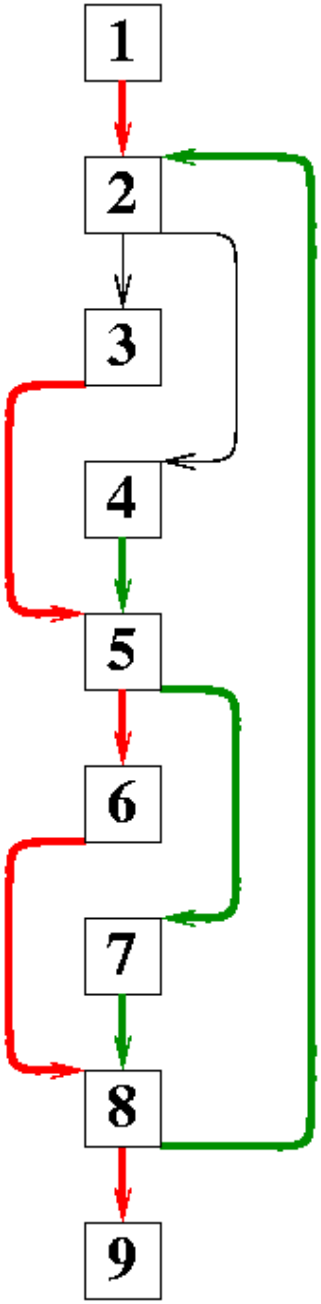
	UB–WCET	LB–WCET	
Path A:	33	24	2 → 3 → 5 → 6 → 8 →
Path B:	34	25	2 → 3 → 5 → 7 → 8 →
Path C:	34	22	2 → 4 → 5 → 6 → 8 →
Path D:	35	26	2 → 4 → 5 → 7 → 8 →
Path E:	31,024	22,024	1 → loop → 9

Example of Positioning Edges (Step 3)



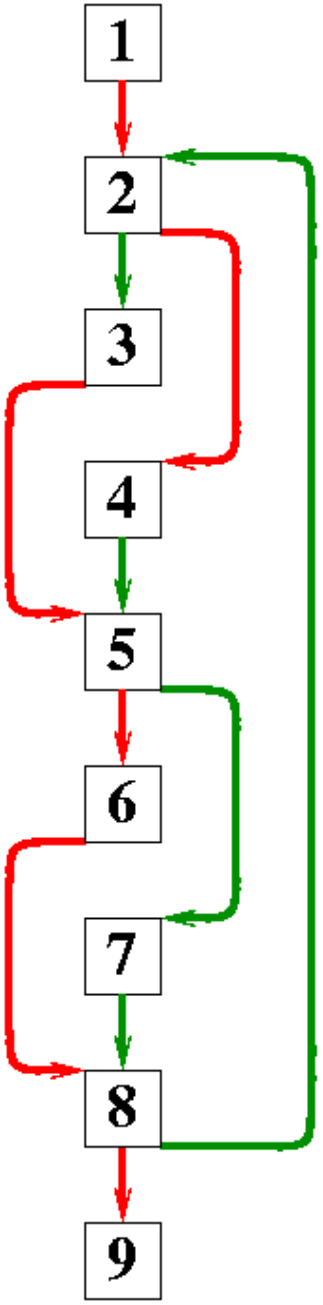
	UB–WCET	LB–WCET	
Path A:	33	27	2 → 3 → 5 → 6 → 8 →
Path B:	31	25	2 → 3 → 5 → 7 → 8 →
Path C:	34	28	2 → 4 → 5 → 6 → 8 →
Path D:	32	26	2 → 4 → 5 → 7 → 8 →
Path E:	30,024	24,024	1 → loop → 9

Example of Positioning Edges (Step 4)



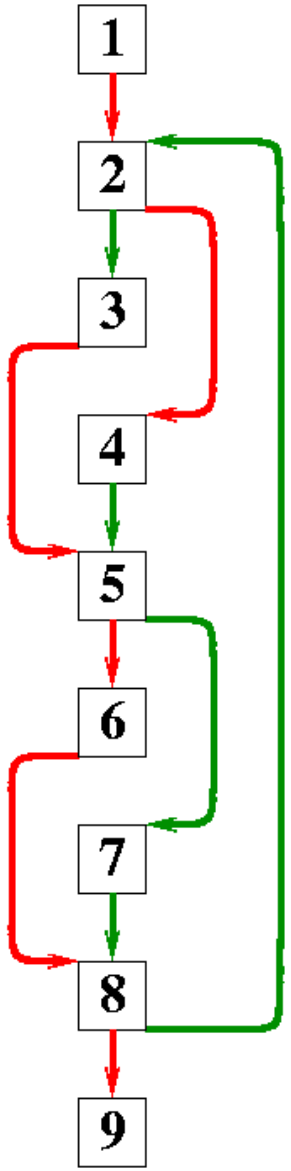
	UB–WCET	LB–WCET	
Path A:	33	30	2 → 3 → 5 → 6 → 8 →
Path B:	31	28	2 → 3 → 5 → 7 → 8 →
Path C:	31	28	2 → 4 → 5 → 6 → 8 →
Path D:	29	26	2 → 4 → 5 → 7 → 8 →
Path E:	29,024	26,024	1 → loop → 9

Example of Positioning Edges (Step 5)

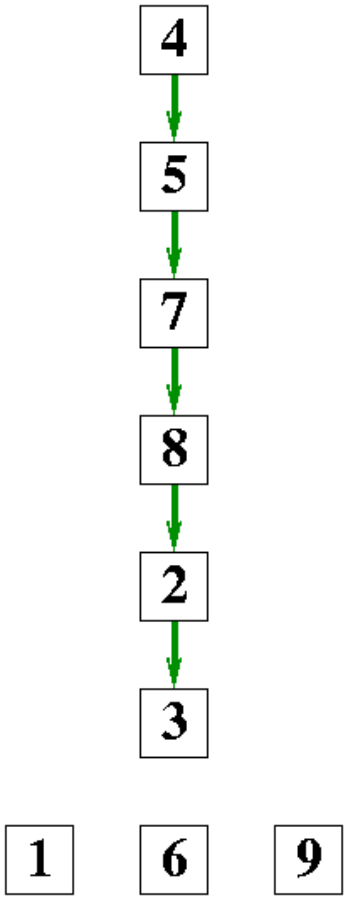


	UB–WCET	LB–WCET	
Path A:	30	30	2 → 3 → 5 → 6 → 8 →
Path B:	28	28	2 → 3 → 5 → 7 → 8 →
Path C:	31	31	2 → 4 → 5 → 6 → 8 →
Path D:	29	29	2 → 4 → 5 → 7 → 8 →
Path E:	27,024	27,024	1 → loop → 9
Initial:	37,020	22,018	

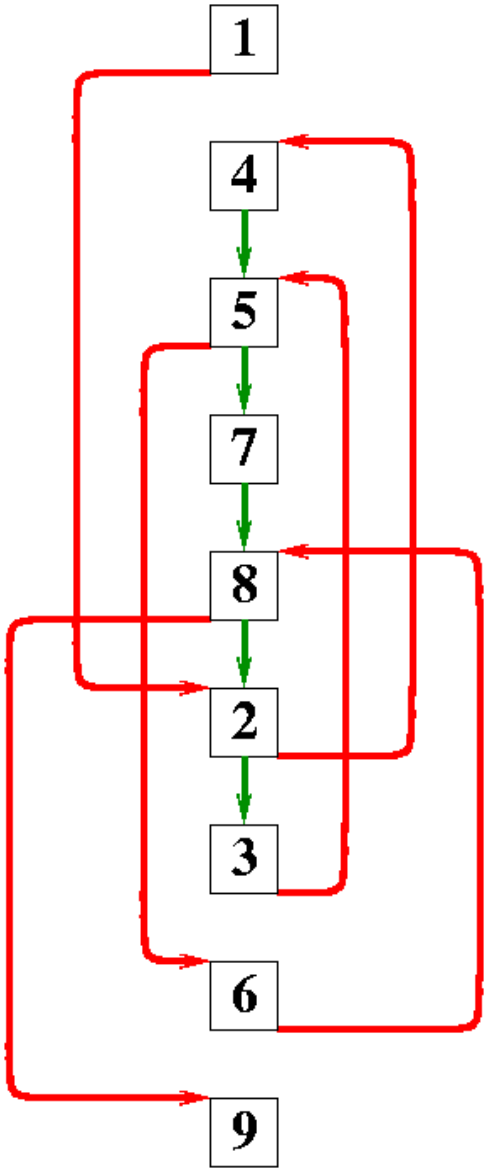
Establishing the Final Positioning



Original
Control Flow



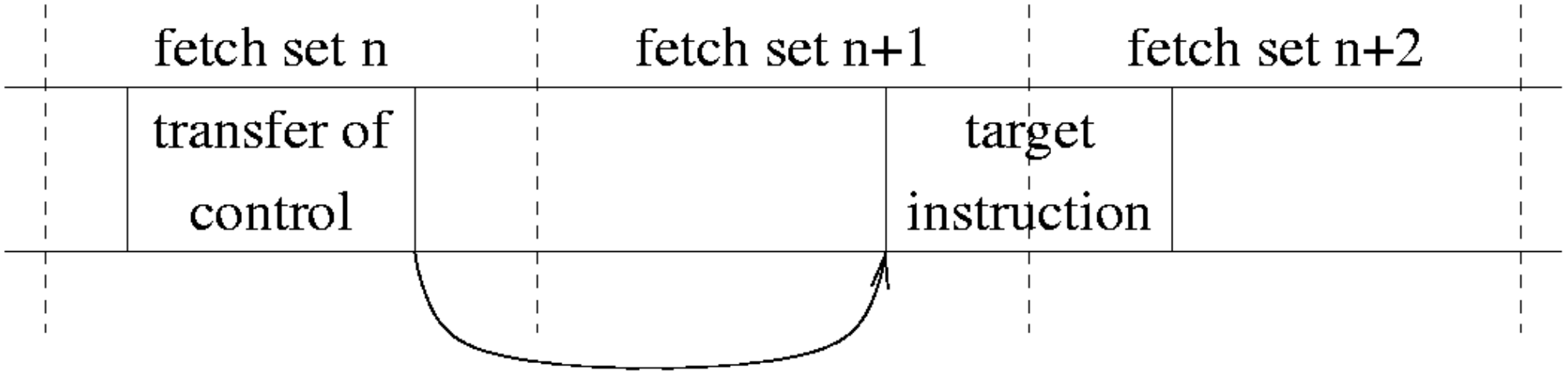
After
Connecting the
Contiguous Edges



Final
Control Flow

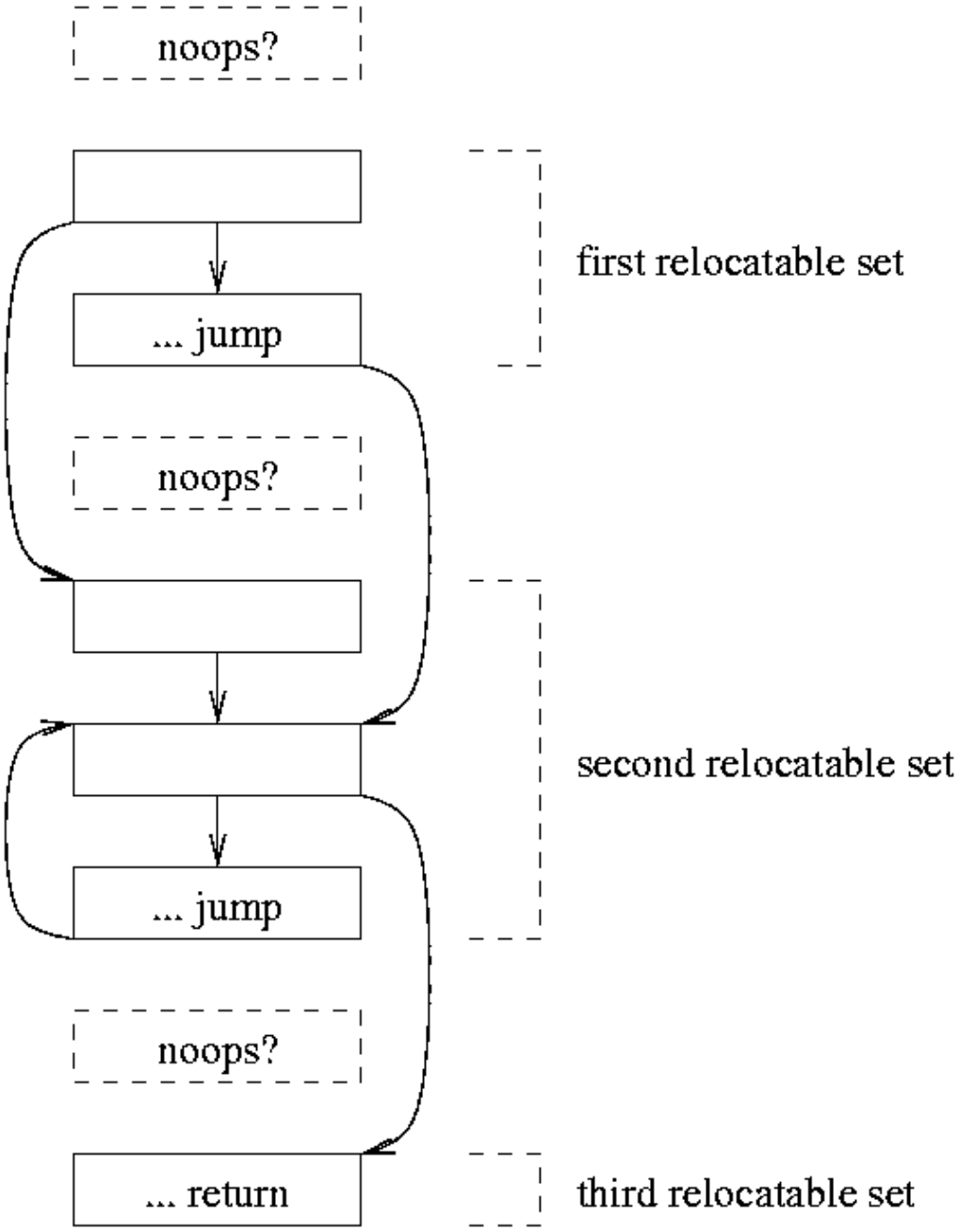
WCET Target Alignment

- SC100 instructions are grouped into aligned fetch sets that are four words (8 bytes) in size.
- When transfer of control occurs to a target instruction that spans more than one fetch set, then the processor stalls for an additional cycle.



WCET Target Alignment Algorithm

- Partition the function into relocatable sets of basic blocks.
- Attempt to align each relocatable set by inserting 0-3 noops and determine which produces the best WCET.



Benchmarks

Program	Description
bubblesort	performs a bubble sort on 500 elements
keysearch	performs a linear search involving 4 nested loops for 625 elements
summidall	sums the middle half and all elements of a 1000 integer vector
summinmax	sums the min and max of corresponding elements of two 1000 integer vectors
sumoddeven	sums the odd and even elements of a 1000 integer vector
sumnegpos	sums the negative, positive, and all elements of a 1000 integer vector
sumposclrneg	sums positive values from two 1000 element arrays and sets negative values to zero
sym	tests if a 100x100 matrix is symmetric
unweight	converts an adjacency 100x100 matrix of a weighted graph to an unweighted graph

Baseline WCET

Program	Observed Cycles	WCET Cycles	WCET Ratio
bubblesort	7,497,532	7,748,545	1.033
keysearch	30,667	31,143	1.016
summidall	19,508	19,515	1.000
summinmax	24,010	24,015	1.000
sumnegpos	20,010	20,015	1.000
sumoddeven	22,021	23,027	1.046
sumposclrneg	31,013	31,018	1.000
sym	223,168	223,472	1.001
unweight	350,507	350,814	1.001
average	913,160	941,285	1.011

After WCET Positioning and Alignment

Program	After WCET Positioning		After WCET Alignment		Time Ratio
	WCET Cycles	Positioning Ratio	WCET Cycles	Alignment Ratio	
bubblesort	7,747,045	**1.000**	7,622,296	**0.984**	1.94
keysearch	29,268	**0.940**	29,268	**0.940**	2.17
summidall	16,721	**0.857**	16,721	**0.857**	1.33
summinmax	22,021	**0.917**	21,023	**0.875**	1.56
sumnegpos	18,021	**0.900**	18,021	**0.900**	1.33
sumoddeven	18,030	**0.783**	16,543	**0.718**	1.67
sumposclrneg	27,024	**0.871**	27,024	**0.871**	1.90
sym	208,622	**0.934**	208,622	**0.934**	1.70
unweight	341,020	**0.972**	341,020	**0.972**	2.00
average	936,419	**0.908**	922,282	**0.895**	1.73

After WCET Positioning and Alignment

Program	After WCET Positioning		After WCET Alignment		Time Ratio
	WCET Cycles	Positioning Ratio	WCET Cycles	Alignment Ratio	
bubblesort	7,747,045	**1.000**	7,622,296	**0.984**	1.94
keysearch	29,268	**0.940**	29,268	**0.940**	2.17
summidall	16,721	**0.857**	16,721	**0.857**	1.33
summinmax	22,021	**0.917**	21,023	**0.875**	1.56
sumnegpos	18,021	**0.900**	18,021	**0.900**	1.33
sumoddeven	18,030	**0.783**	16,543	**0.718**	1.67
sumposclrneg	27,024	**0.871**	27,024	**0.871**	1.90
sym	208,622	**0.934**	208,622	**0.934**	1.70
unweight	341,020	**0.972**	341,020	**0.972**	2.00
average	936,419	**0.908**	922,282	**0.895**	1.73

After WCET Positioning and Alignment

Program	After WCET Positioning		After WCET Alignment		Time Ratio
	WCET Cycles	Positioning Ratio	WCET Cycles	Alignment Ratio	
bubblesort	7,747,045	**1.000**	7,622,296	**0.984**	1.94
keysearch	29,268	**0.940**	29,268	**0.940**	2.17
summidall	16,721	**0.857**	16,721	**0.857**	1.33
summinmax	22,021	**0.917**	21,023	**0.875**	1.56
sumnegpos	18,021	**0.900**	18,021	**0.900**	1.33
sumoddeven	18,030	**0.783**	16,543	**0.718**	1.67
sumposclrneg	27,024	**0.871**	27,024	**0.871**	1.90
sym	208,622	**0.934**	208,622	**0.934**	1.70
unweight	341,020	**0.972**	341,020	**0.972**	2.00
average	936,419	**0.908**	922,282	**0.895**	1.73

After WCET Positioning and Alignment

Program	After WCET Positioning		After WCET Alignment		Time Ratio
	WCET Cycles	Positioning Ratio	WCET Cycles	Alignment Ratio	
bubblesort	7,747,045	**1.000**	7,622,296	**0.984**	1.94
keysearch	29,268	**0.940**	29,268	**0.940**	2.17
summidall	16,721	**0.857**	16,721	**0.857**	1.33
summinmax	22,021	**0.917**	21,023	**0.875**	1.56
sumnegpos	18,021	**0.900**	18,021	**0.900**	1.33
sumoddeven	18,030	**0.783**	16,543	**0.718**	1.67
sumposclrneg	27,024	**0.871**	27,024	**0.871**	1.90
sym	208,622	**0.934**	208,622	**0.934**	1.70
unweight	341,020	**0.972**	341,020	**0.972**	2.00
average	936,419	**0.908**	922,282	**0.895**	1.73

Greedy vs. Exhaustive Results

Program	Permutations	Ratio to Minimum		
		Greedy	Default	Maximum
bubblesort	40,328	**1.000**	1.000	1.194
keysearch	39,916,801	**1.000**	1.064	2.038
summidall	5,040	**1.000**	1.107	1.718
summinmax	362,880	**1.000**	1.142	1.428
sumnegpos	5,040	**1.000**	1.111	1.333
sumoddeven	3,628,800	**1.000**	1.375	1.937
sumposclrneg	362,880	**1.000**	1.148	1.370
sym	5041	**1.000**	1.071	1.145
unweight	40,320	**1.000**	1.028	1.353
average	5,540,851	**1.000**	1.116	1.502

Greedy vs. Exhaustive Results

Program	Permutations	Ratio to Minimum		
		Greedy	Default	Maximum
bubblesort	40,328	**1.000**	1.000	1.194
keysearch	39,916,801	**1.000**	1.064	2.038
summidall	5,040	**1.000**	1.107	1.718
summinmax	362,880	**1.000**	1.142	1.428
sumnegpos	5,040	**1.000**	1.111	1.333
sumoddeven	3,628,800	**1.000**	1.375	1.937
sumposclrneg	362,880	**1.000**	1.148	1.370
sym	5041	**1.000**	1.071	1.145
unweight	40,320	**1.000**	1.028	1.353
average	5,540,851	**1.000**	1.116	1.502

Future Work

- Attempt to prove that our WCET code positioning algorithm is optimal.
- Test with larger benchmarks to see the impact on WCET.
- Adapt other compiler optimizations to improve WCET.

Conclusions

- Developed a WCET code positioning algorithm driven by WCET path information.
  - Assume all edges are initially unpositioned.
  - Use WCET path information to select the next edge to make contiguous.
  - Interact with a timing analyzer to get up-to-date information.
  - Showed our greedy algorithm obtains optimal results for our benchmark suite.
- Developed a WCET target alignment optimization.
- Reduced WCET by 10.5% on average.