

The Hot Path SSA Form: Algorithms and Applications

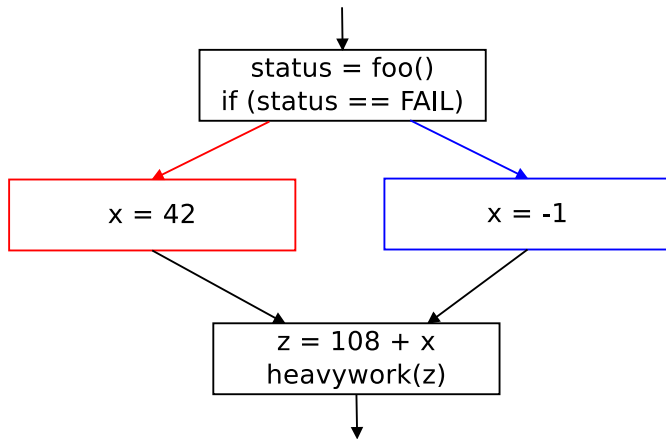
Subhajit Roy

*Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur*

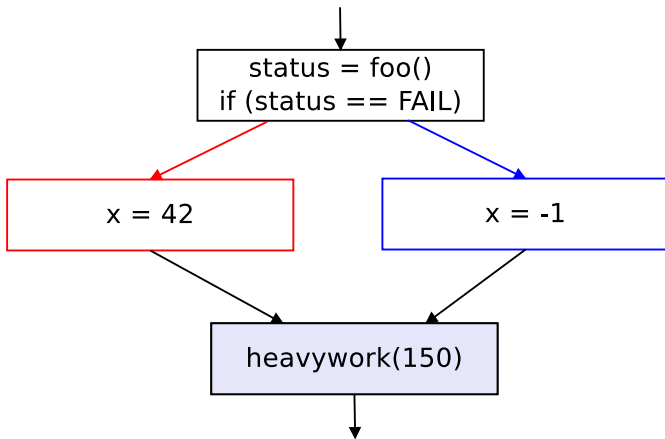
Outline

- 1 Introduction
- 2 The Hot Path SSA (HPSSA) Form
- 3 Constructing the HPSSA Form
- 4 Conclusions

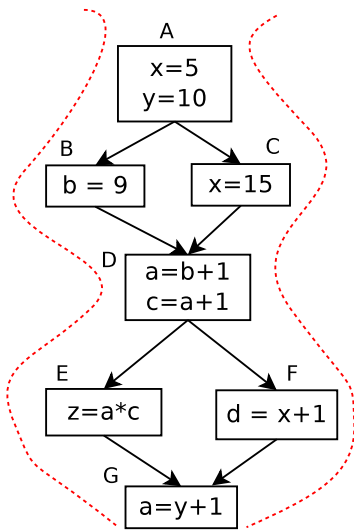
A case for profile-guided optimizations (PGO)



A case for profile-guided optimizations (PGO)



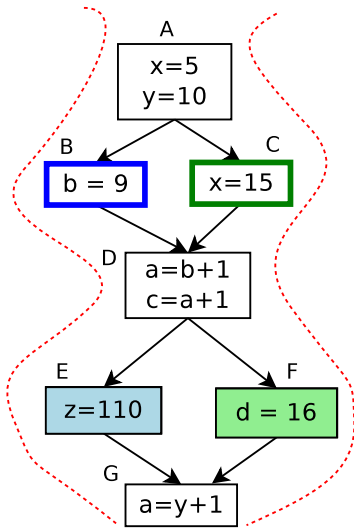
Profile-guided analysis on paths



Summary

- Profile-guided analysis across paths is stronger—can capture correlations between control-flow of basic-blocks
- Collecting path-profiles seems challenging—requires “recording” of a sequence of basic-blocks

Profile-guided analysis on paths



Summary

- Profile-guided analysis across paths is stronger—can capture correlations between control-flow of basic-blocks
- Collecting path-profiles seems challenging—requires “recording” of a sequence of basic-blocks

Profiling acyclic paths

Ball-Larus Acyclic Profiling [Ball & Larus, MICRO'96]

- Core idea: assign an identifier to each path, that can be calculated efficiently at runtime
- Record frequencies against these identifiers (instead of a sequence of node identifiers)

Profiling acyclic paths

Ball-Larus Acyclic Profiling [Ball & Larus, MICRO'96]

- Core idea: assign an identifier to each path, that can be calculated efficiently at runtime
- Record frequencies against these identifiers (instead of a sequence of node identifiers)

Capturing still longer paths (k-iteration paths)

- Allows capturing correlations across loop iterations [Roy & Srikant, CGO'09]; a generalization of the Ball-Larus algorithm
- Subsequent work by other groups [D'Elia & Demetrescu, OOPSLA'13]; uses a prefix forest to record BL paths

Profile-guided analyses

- **Code understanding**
 - Can expose refactoring opportunities

Profile-guided analyses

- **Code understanding**
 - Can expose refactoring opportunities
- **Program testing and verification**
 - Data-driven synthesis of invariants
 - Guided testing for low frequency paths

Profile-guided analyses

- **Code understanding**
 - Can expose refactoring opportunities
- **Program testing and verification**
 - Data-driven synthesis of invariants
 - Guided testing for low frequency paths
- **Profile-guided optimizations**

Profile-guided analyses to optimizations

- **Speculation:** *check and retry*
 - eg. value speculation
 - Can impact code size (significant impact w/o speculation support in hardware)

Profile-guided analyses to optimizations

- **Speculation:** *check and retry*
 - eg. value speculation
 - Can impact code size (significant impact w/o speculation support in hardware)
- **Compensation code**
 - eg. superblock scheduling
 - Can impact code size

Profile-guided analyses to optimizations

- **Speculation:** *check and retry*
 - eg. value speculation
 - Can impact code size (significant impact w/o speculation support in hardware)
- **Compensation code**
 - eg. superblock scheduling
 - Can impact code size
- **Error resilient modules**
 - modules for statistical summarization on samples, generate data for ML models
 - No impact on code size

Profile-guided analyses to optimizations

- **Speculation:** *check and retry*
 - eg. value speculation
 - Can impact code size (significant impact w/o speculation support in hardware)
- **Compensation code**
 - eg. superblock scheduling
 - Can impact code size
- **Error resilient modules**
 - modules for statistical summarization on samples, generate data for ML models
 - No impact on code size
- **Optimizations that don't impact correctness**
 - eg. register allocation
 - No impact on code size

Outline

- 1 Introduction
- 2 The Hot Path SSA (HPSSA) Form
- 3 Constructing the HPSSA Form
- 4 Conclusions

Why is path-profile-guided analysis hard?

disparate data-structures: program + profile

```
int main ( )
{
    int a[10] = {(0,2,4), (1,2,4), (7,5,4), (2,5,1)};
    for (int i=0; i<4; i++)
    {
        int max=0;
        int x = a[i][0];
        int y = a[i][1];
        int z = a[i][2];
        int t1 = ( x >= y ) ;
        int t2 = ( y >= z ) ;
        int t3 = ( z >= x ) ; // loop
        printf("t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16,t17,t18,t19,t20,t21,t22,t23,t24,t25,t26,t27,t28,t29,t30,t31,t32,t33,t34,t35,t36,t37,t38,t39,t40,t41,t42,t43,t44,t45,t46,t47,t48,t49,t50,t51,t52,t53,t54,t55,t56,t57,t58,t59,t60,t61,t62,t63,t64,t65,t66,t67,t68,t69,t70,t71,t72,t73,t74,t75,t76,t77,t78,t79,t80,t81,t82,t83,t84,t85,t86,t87,t88,t89,t90,t91,t92,t93,t94,t95,t96,t97,t98,t99,t100,t101,t102,t103,t104,t105,t106,t107,t108,t109,t110,t111,t112,t113,t114,t115,t116,t117,t118,t119,t120,t121,t122,t123,t124,t125,t126,t127,t128,t129,t130,t131,t132,t133,t134,t135,t136,t137,t138,t139,t140,t141,t142,t143,t144,t145,t146,t147,t148,t149,t150,t151,t152,t153,t154,t155,t156,t157,t158,t159,t160,t161,t162,t163,t164,t165,t166,t167,t168,t169,t170,t171,t172,t173,t174,t175,t176,t177,t178,t179,t180,t181,t182,t183,t184,t185,t186,t187,t188,t189,t190,t191,t192,t193,t194,t195,t196,t197,t198,t199,t200,t201,t202,t203,t204,t205,t206,t207,t208,t209,t210,t211,t212,t213,t214,t215,t216,t217,t218,t219,t220,t221,t222,t223,t224,t225,t226,t227,t228,t229,t230,t231,t232,t233,t234,t235,t236,t237,t238,t239,t240,t241,t242,t243,t244,t245,t246,t247,t248,t249,t250,t251,t252,t253,t254,t255,t256,t257,t258,t259,t260,t261,t262,t263,t264,t265,t266,t267,t268,t269,t270,t271,t272,t273,t274,t275,t276,t277,t278,t279,t280,t281,t282,t283,t284,t285,t286,t287,t288,t289,t290,t291,t292,t293,t294,t295,t296,t297,t298,t299,t300,t301,t302,t303,t304,t305,t306,t307,t308,t309,t310,t311,t312,t313,t314,t315,t316,t317,t318,t319,t320,t321,t322,t323,t324,t325,t326,t327,t328,t329,t330,t331,t332,t333,t334,t335,t336,t337,t338,t339,t340,t341,t342,t343,t344,t345,t346,t347,t348,t349,t350,t351,t352,t353,t354,t355,t356,t357,t358,t359,t360,t361,t362,t363,t364,t365,t366,t367,t368,t369,t370,t371,t372,t373,t374,t375,t376,t377,t378,t379,t380,t381,t382,t383,t384,t385,t386,t387,t388,t389,t390,t391,t392,t393,t394,t395,t396,t397,t398,t399,t400,t401,t402,t403,t404,t405,t406,t407,t408,t409,t410,t411,t412,t413,t414,t415,t416,t417,t418,t419,t420,t421,t422,t423,t424,t425,t426,t427,t428,t429,t430,t431,t432,t433,t434,t435,t436,t437,t438,t439,t440,t441,t442,t443,t444,t445,t446,t447,t448,t449,t450,t451,t452,t453,t454,t455,t456,t457,t458,t459,t460,t461,t462,t463,t464,t465,t466,t467,t468,t469,t470,t471,t472,t473,t474,t475,t476,t477,t478,t479,t480,t481,t482,t483,t484,t485,t486,t487,t488,t489,t490,t491,t492,t493,t494,t495,t496,t497,t498,t499,t500,t501,t502,t503,t504,t505,t506,t507,t508,t509,t510,t511,t512,t513,t514,t515,t516,t517,t518,t519,t520,t521,t522,t523,t524,t525,t526,t527,t528,t529,t530,t531,t532,t533,t534,t535,t536,t537,t538,t539,t540,t541,t542,t543,t544,t545,t546,t547,t548,t549,t550,t551,t552,t553,t554,t555,t556,t557,t558,t559,t560,t561,t562,t563,t564,t565,t566,t567,t568,t569,t570,t571,t572,t573,t574,t575,t576,t577,t578,t579,t580,t581,t582,t583,t584,t585,t586,t587,t588,t589,t590,t591,t592,t593,t594,t595,t596,t597,t598,t599,t600,t601,t602,t603,t604,t605,t606,t607,t608,t609,t610,t611,t612,t613,t614,t615,t616,t617,t618,t619,t620,t621,t622,t623,t624,t625,t626,t627,t628,t629,t630,t631,t632,t633,t634,t635,t636,t637,t638,t639,t640,t641,t642,t643,t644,t645,t646,t647,t648,t649,t650,t651,t652,t653,t654,t655,t656,t657,t658,t659,t660,t661,t662,t663,t664,t665,t666,t667,t668,t669,t670,t671,t672,t673,t674,t675,t676,t677,t678,t679,t680,t681,t682,t683,t684,t685,t686,t687,t688,t689,t690,t691,t692,t693,t694,t695,t696,t697,t698,t699,t700,t701,t702,t703,t704,t705,t706,t707,t708,t709,t710,t711,t712,t713,t714,t715,t716,t717,t718,t719,t720,t721,t722,t723,t724,t725,t726,t727,t728,t729,t730,t731,t732,t733,t734,t735,t736,t737,t738,t739,t740,t741,t742,t743,t744,t745,t746,t747,t748,t749,t750,t751,t752,t753,t754,t755,t756,t757,t758,t759,t760,t761,t762,t763,t764,t765,t766,t767,t768,t769,t770,t771,t772,t773,t774,t775,t776,t777,t778,t779,t780,t781,t782,t783,t784,t785,t786,t787,t788,t789,t790,t791,t792,t793,t794,t795,t796,t797,t798,t799,t800,t801,t802,t803,t804,t805,t806,t807,t808,t809,t810,t811,t812,t813,t814,t815,t816,t817,t818,t819,t820,t821,t822,t823,t824,t825,t826,t827,t828,t829,t830,t831,t832,t833,t834,t835,t836,t837,t838,t839,t840,t841,t842,t843,t844,t845,t846,t847,t848,t849,t850,t851,t852,t853,t854,t855,t856,t857,t858,t859,t860,t861,t862,t863,t864,t865,t866,t867,t868,t869,t870,t871,t872,t873,t874,t875,t876,t877,t878,t879,t880,t881,t882,t883,t884,t885,t886,t887,t888,t889,t890,t891,t892,t893,t894,t895,t896,t897,t898,t899,t900,t901,t902,t903,t904,t905,t906,t907,t908,t909,t910,t911,t912,t913,t914,t915,t916,t917,t918,t919,t920,t921,t922,t923,t924,t925,t926,t927,t928,t929,t930,t931,t932,t933,t934,t935,t936,t937,t938,t939,t940,t941,t942,t943,t944,t945,t946,t947,t948,t949,t950,t951,t952,t953,t954,t955,t956,t957,t958,t959,t960,t961,t962,t963,t964,t965,t966,t967,t968,t969,t970,t971,t972,t973,t974,t975,t976,t977,t978,t979,t980,t981,t982,t983,t984,t985,t986,t987,t988,t989,t990,t991,t992,t993,t994,t995,t996,t997,t998,t999,1000);
        if ( t1 ) max = z;
        if ( t1 && t2 ) max = x;
        if ( t1 && t3 ) max = x;
        if ( t2 && t1 ) max = y;
        printf("20\n", max);
    }
}
```

1-2-4-5	30
21-2-5-5-1	25
1-2-4-5	30
21-2-5-5-1	25
1-2-4-5	30
21-2-5-5-1	25
1-2-4-5	30
21-2-5-5-1	25

Why is path-profile-guided analysis hard?

- There has been enough interest in path-profile-guided analysis and optimizations....
- ...however, designing path-profile-guided variants of traditional optimizations remained hard
- ...hard enough to justify *publications per optimization*
 - Gupta, Benson, Fang. Path profile guided partial dead code elimination using predication. PACT '97.
 - Gupta, Benson, Fang. Path profile guided partial redundancy elimination using speculation. ICCL '98.
 - ...

Our Objective

Can we **weave** profile information into the program representation

Our Objective

Can we **weave** profile information into the program representation
....into a **single, consistent** data-structure

Our Objective

Can we **weave** profile information into the program representation

....into a **single, consistent** data-structure

... that provides the convenience and elegance of an **SSA-like** intermediate form

Our Objective

Can we **weave** profile information into the program representation

....into a **single, consistent** data-structure

... that provides the convenience and elegance of an **SSA-like** intermediate form

...allowing the design of profile-guided versions of “traditional” optimizations with
trivial algorithmic modification of the base algorithms

Our Objective

Can we **weave** profile information into the program representation

....into a **single, consistent** data-structure

... that provides the convenience and elegance of an **SSA-like** intermediate form

...allowing the design of profile-guided versions of “traditional” optimizations with
trivial algorithmic modification of the base algorithms

...providing a **common representation** for both “traditional” as well as profile-guided
analysis and optimizations

The Hot Path SSA Form (HPSSA)

Semantics of a ϕ -function

$$y = \phi(x_1, x_2, \dots, x_n)$$

The Hot Path SSA Form (HPSSA)

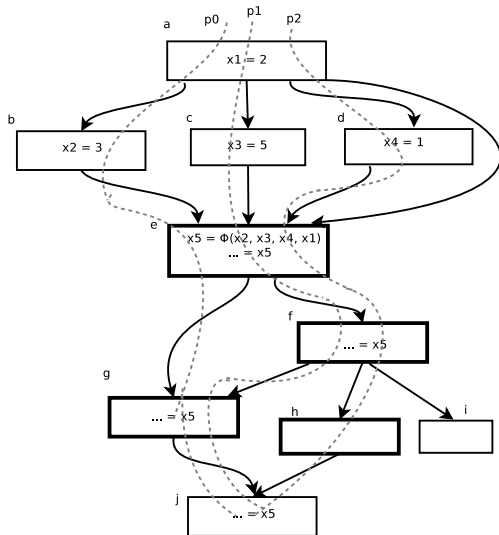
Semantics of a ϕ -function

$$y = \phi(x_1, x_2, \dots, x_n)$$

Semantics of a τ -function

$$\tau(x_0, x_1, x_2, \dots, x_n) = \begin{cases} x_0 & \text{safe interp.} \\ \phi(x_1, x_2, \dots, x_n) & \text{speculative interp.} \end{cases}$$

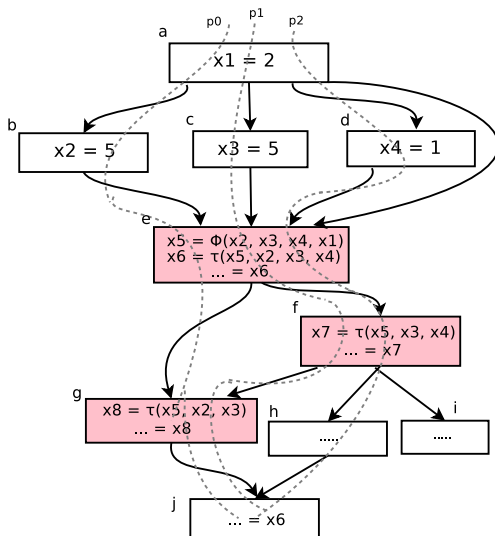
The SSA form



No frequent path carrying:

- def $x_2 = 3$ to use at block **f**
- def $x_4 = 1$ to use at block **g**
- def $x_1 = 2$ to either **f** or **g**

The Hot Path SSA Form



No frequent path carrying:

- def $x_2 = 3$ to use at block **f**
- def $x_4 = 1$ to use at block **g**

The Hot Path SSA Form

Properties

If a program is in the Hot Path SSA form, then,

- each use of a variable is reachable by a single definition; [SSA-like form]

The Hot Path SSA Form

Properties

If a program is in the Hot Path SSA form, then,

- each use of a variable is reachable by a single definition; [SSA-like form]
- **safe interpretation:** [supports traditional analysis]
 - each use of a variable is reachable by the *meet-over-all-paths* reaching definition chains;

The Hot Path SSA Form

Properties

If a program is in the Hot Path SSA form, then,

- each use of a variable is reachable by a single definition; [SSA-like form]
- **safe interpretation:** [supports traditional analysis]
 - each use of a variable is reachable by the *meet-over-all-paths* reaching definition chains;
- **speculative interpretation:** [supports profile-guided analysis]
 - each use of a variable in a basic-block is reachable by the *meet-over-frequent-paths* reaching definitions. ^a

^aor the meet-over-all-paths reaching definition chains, if the use is not reachable from any meet-over-hot-paths reaching definition chain

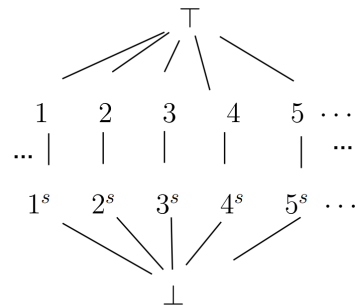
Speculative Sparse Conditional Constant Propagation (SSCCP)

- Introduce new speculative values $\{\dots, 1^s, 2^s, \dots\} \in C^s$
- Operation with *speculative* values result in *speculative* results (with same semantics as base operator)

$$\alpha^s \langle op \rangle \beta = (\alpha \langle op \rangle \beta)^s$$

- Transfer function for τ -functions ($\beta = x_1 \sqcup x_2 \sqcup \dots$)

$$\tau(x_0, x_1, \dots, x_n) \sqcup \begin{cases} x_0 \sqcup \top & \text{if } x_0 \sqcup \beta \neq \top \\ \beta & \text{if } x_0 \sqcup \beta = \top \wedge \beta \in C^s \\ \beta^s & \text{otherwise} \end{cases}$$



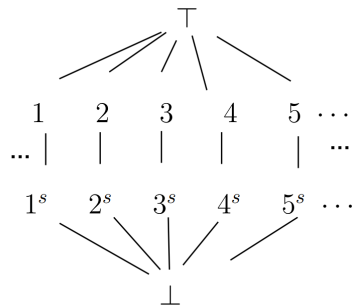
Speculative Sparse Conditional Constant Propagation (SSCCP)

- Introduce new speculative values $\{\dots, 1^s, 2^s, \dots\} \in C^s$
- Operation with *speculative* values result in *speculative* results (with same semantics as base operator)

$$\alpha^s \langle op \rangle \beta = (\alpha \langle op \rangle \beta)^s$$

- Transfer function for τ -functions ($\beta = x_1 \sqcup x_2 \sqcup \dots$)

$$\tau(x_0, x_1, \dots, x_n) \sqcup \begin{cases} x_0 \sqcup \top & \text{if } x_0 \sqcup \beta \neq \top \\ \beta & \text{if } x_0 \sqcup \beta = \top \wedge \beta \in C^s \\ \beta^s & \text{otherwise} \end{cases}$$



Almost trivial to generate profile-guided variants of standard analyses—it took us an afternoon to “port” SCCP to SSCCP!

Speculative Sparse Conditional Constant Propagation (SSCCP)

Table 1. Speculative Constants discovered by the SSCP algorithm. (‘~’ indicates *almost*; *grp*, *prg*, & *src* refer to inputs *graphic*, *program* & *source* respectively).

Program	Inpt	Variable Uses		Expression Uses		Total		
		Uses	HitRt	Uses	HitRt	Hits	Misses	HitRt
<i>181.mcf</i>	-	33110	100.00	49665	100.00	82775	0	100.00
<i>175.vpr</i>	-	6938074	100.00	8110837	100.00	15048911	0	100.00
<i>164.gzip</i>	grp	26592	100.00	5	100.00	26597	0	100.00
	prg	17412	100.00	5	100.00	17417	0	100.00
	src	4721	99.98	5	100.00	4725	1	99.98
<i>197.parser</i>	-	165970964	~100.00	340	97.94	165970861	443	~100.00
<i>256.bzip2</i>	grp	132106650	~100.00	938	76.97	132107372	216	~100.00
	prg	100819492	~100.00	6576416	15.67	101849942	5545966	94.84
	src	108134316	~100.00	5256006	17.94	109077366	4312956	96.20

Speculative Sparse Conditional Constant Propagation (SSCCP)

Table 1. Speculative Constants discovered by the SSCP algorithm. (‘~’ indicates *almost*; *grp*, *prg*, & *src* refer to inputs *graphic*, *program* & *source* respectively).

Program	Inpt	Variable Uses		Expression Uses		Total		
		Uses	HitRt	Uses	HitRt	Hits	Misses	HitRt
<i>181.mcf</i>	-	33110	100.00	49665	100.00	82775	0	100.00
<i>175.vpr</i>	-	6938074	100.00	8110837	100.00	15048911	0	100.00
<i>164.gzip</i>	grp	26592	100.00	5	100.00	26597	0	100.00
	prg	17412	100.00	5	100.00	17417	0	100.00
	src	4721	99.98	5	100.00	4725	1	99.98
<i>197.parser</i>	-	165970964	~100.00	340	97.94	165970861	443	~100.00
<i>256.bzip2</i>	grp	132106650	~100.00	938	76.97	132107372	216	~100.00
	prg	100819492	~100.00	6576416	15.67	101849942	5545966	94.84
	src	108134316	~100.00	5256006	17.94	109077366	4312956	96.20

Optimized on “train” inputs, results produced on “ref” inputs... Shows the value of profile-guided analysis for code understanding and optimizations...

Profile-Guided Register Allocation [Jain et al., HiPC'16]

<pre># BB#4: addl %edi, 32(%esp) addl \$20, %esi addl \$117, %ebp jmp .LBB0_6 .align 16, 0x90 .LBB0_5: addl \$101, 40(%esp) subl 40(%esp), %ecx addl \$33, %edi addl \$-11, %ebp</pre>	<pre># %if.then # in Loop: Header=B # 4-byte Folded Spill # %if.else # in Loop: Header=B # 4-byte Folded Spill # 4-byte Folded Reload > ></pre>	<pre># BB#4: addl %edi, %ecx addl \$20, %esi addl \$117, %ebp jmp .LBB0_6 .align 16, 0x90 .LBB0_5: addl \$101, 40(%esp) movl 36(%esp), %edx subl 40(%esp), %edx movl %edx, 36(%esp) addl \$33, %edi addl \$-11, %ebp</pre>	<pre># %if.then # in Loop: Header=B # %if.else # in Loop: Header=B # 4-byte Folded Spill # 4-byte Reload # 4-byte Folded Reload # 4-byte Spill</pre>
--	---	--	--

.LBB0_5 is hot: 40(%esp) is cached in %ecx	BB#4 is hot: 32(%esp) is cached in %ecx
--	---

ILP based register allocator

- Implements integer linear programming based profile-guided register allocation
- Base allocator (without profile information) is simplified version of [Goodwin and Wilken, 1996]

Base Allocator

min $\sum_{v \in V, l \in L} S_{v,l} \cdot x_{v,l,\sigma}$, (minimize spills) *such that*

$\forall v \in V \forall l \in L \sum_{r \in R} x_{v,l,r} = 1$ (a variable to at least one location)

$\forall r \in R - \sigma \forall l \in L \sum_{v \in V} x_{v,l,r} \leq 1$ (a register holds at most one variable)

Cost for τ -functions

Modelling τ -functions

The cost for $t = \tau(\text{safe}, t_1, t_2, \dots, t_n)$:

$$\text{TauCost} = \sum_{i=1}^{i=n} (\text{loc}(t, t_i) * \text{freq}(t_i))$$

where $\text{loc}(t, t_i)$ is 1 for the locations where t and t_i are different, else 0.

The objective function

The objective function for register allocation over HPSSA is modified to include the *TauCost* discussed above.

The objective function

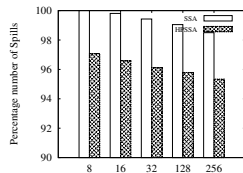
$$\min \sum_{v \in V, l \in L} S_{v,l} \cdot x_{v,l,\sigma} + \sum_{i=1}^{i=m} \textit{TauCost}_i \quad (1)$$

where, $\textit{TauCost}_i$ is the cost for the i^{th} τ -function (and m is the total number of τ -functions in the program), and

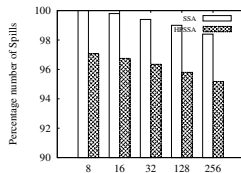
$$S_{v,l} = \begin{cases} 100, & \text{if } v \text{ has further uses along hot path;} \\ 15, & \text{if } v \text{ has further uses only along cold path;} \\ 0, & \text{if } v \text{ has no further use.} \end{cases}$$

Almost a trivial modification to the objective function...

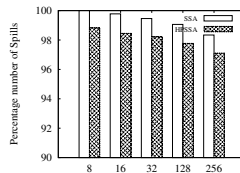
Percentage of spills for different register sizes



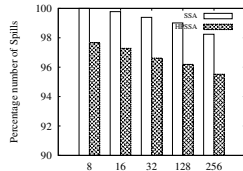
(a) gzip



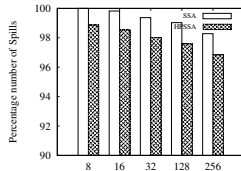
(b) bzip2



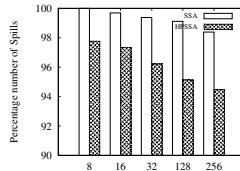
(c) mcf



(d) Crafty



(e) GAP



(f) Parser

Outline

- 1 Introduction
- 2 The Hot Path SSA (HPSSA) Form
- 3 Constructing the HPSSA Form**
- 4 Conclusions

Hot/Cold Paths

Definition 1. Hot/Cold Paths

A program path $p : n_1 \rightsquigarrow n_2$ is said to be hot (cold) if the sequence of edges from node n_1 to n_2 appears (does not appear) in any profiled path that occurs frequently in the program profile.

Hot/Cold Paths

Definition 1. Hot/Cold Paths

A program path $p : n_1 \rightsquigarrow n_2$ is said to be hot (cold) if the sequence of edges from node n_1 to n_2 appears (does not appear) in any profiled path that occurs frequently in the program profile.

Definition 2. Temperature (θ) of a node (edge)

- hot: if the node (edge) is present on a hot path;
- cold: if the node (edge) is not present on any hot path.

Hot/Cold Paths

Definition 1. Hot/Cold Paths

A program path $p : n_1 \rightsquigarrow n_2$ is said to be hot (cold) if the sequence of edges from node n_1 to n_2 appears (does not appear) in any profiled path that occurs frequently in the program profile.

Definition 2. Temperature (θ) of a node (edge)

- hot: if the node (edge) is present on a hot path;
- cold: if the node (edge) is not present on any hot path.

Definition 3. Hot/Cold Reaching Definitions and Definition Chains

A definition δ at a basic-block n_1 is said to reach a respective use at a basic-block n_2 hot if there exists a hot path from n_1 to n_2 , and δ is not killed along that path. A definition δ at a basic-block n_1 is said to reach a respective use at a basic-block n_2 cold if there does not exist a hot path from n_1 to n_2 , and δ is not killed at least along one cold path from n_1 to n_2 .

Inserting τ – functions

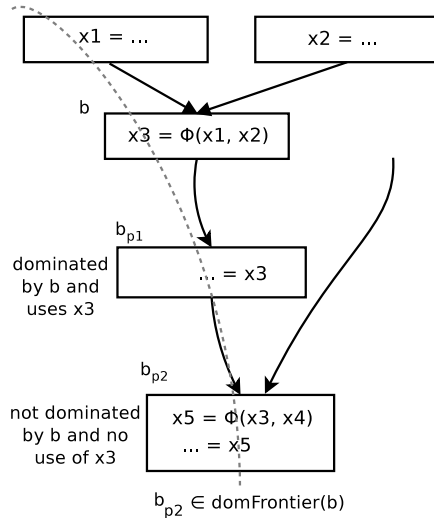
Necessary condition for τ -functions

Lemma 1. A node n requires a τ -function for variable x due to a definition d^x (of a variable x) if

- 1 n is the junction of a hot and a cold path, i.e., paths at different temperatures meet at this node;
- 2 n is reachable by at least two different definitions of the variable x .

Proof. If condition I fails, a τ -function is unnecessary as n can then be reachable by only hot or only cold definitions of x . If condition II fails, a τ -function is again unnecessary as the node is then dominated by a definition of x .

Inserting τ -functions



Inserting τ – *functions*

Definition 4. Thermal Frontier (TF)

For definition d defined at a node u reaching v , v is in the Thermal Frontier of (u, d) , $v \in TF(u, d)$, iff

- 1 the node v is also exposed to a reaching definition d' defined at a node $u \notin Dom(w)$ (w not dominated by u)
- 2 $\theta(u \rightsquigarrow v) \neq \theta(w \rightsquigarrow v)$
- 3 v is the first node on the paths $u \rightsquigarrow v$ and $w \rightsquigarrow v$ that satisfies the above properties.

Inserting τ – *functions*

Definition 4. Thermal Frontier (TF)

For definition d defined at a node u reaching v , v is in the Thermal Frontier of (u, d) , $v \in TF(u, d)$, iff

- ① the node v is also exposed to a reaching definition d' defined at a node $u \notin Dom(w)$ (w not dominated by u)
- ② $\theta(u \rightsquigarrow v) \neq \theta(w \rightsquigarrow v)$
- ③ v is the first node on the paths $u \rightsquigarrow v$ and $w \rightsquigarrow v$ that satisfies the above properties.

Theorem

For a set of visible definitions of a variable x at a set of nodes κ , τ -statements are only required at the Iterated Thermal Frontier ITF^x for variable x .

HPSSA construction [Roy et al., CC'10]

- **Insert ϕ -functions:**
 - insert ϕ -functions at the *iterated dominance frontiers*
- **Insert τ -functions**
 - insert τ -functions at the *iterated thermal frontiers*
- **Allocate arguments to ϕ -functions**
 - use a variable stack to allocate the ϕ -function arguments
- **Allocate arguments to τ -functions**
 - maintain path-sensitive reaching definitions for each program variable corresponding to each hot path on a path-sensitive stack;
 - for each instruction in the program, the algorithm update the respective stack to record the change in the path-sensitive reaching definitions due to the instruction;
 - when a τ -function is encountered, the current set of reaching definitions on the stack is used to allocate the speculative arguments for the τ -function.

HPSSA construction over SSA [Jaiswal et. al., SCOPES'17]

Difficulties

- Needs the SSA construction identified, broken into and retrofitted with the HPSSA construction phases.
- The algorithm is quite complex!

HPSSA construction over SSA [Jaiswal et. al., SCOPES'17]

Difficulties

- Needs the SSA construction identified, broken into and retrofitted with the HPSSA construction phases.
- The algorithm is quite complex!

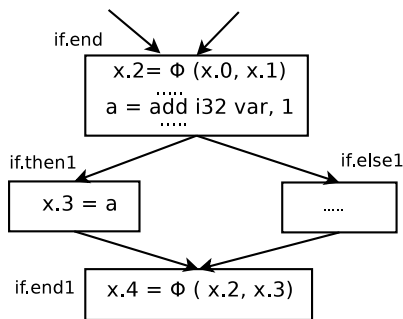
HPSSA over SSA

- **Easily incorporated within existing compilers:** Construction over the SSA form
- **Efficient:** Lesser instructions have to be traversed
- **Simpler:** many constructs are eliminated

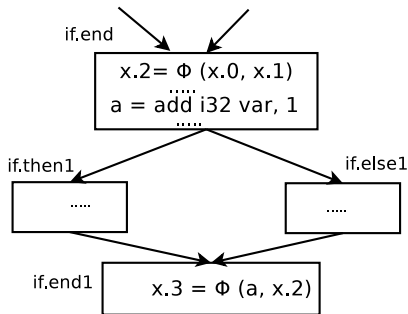
A naïve attempt: Mimic [Roy et al.]

- attempt to “recover” the renamed versions of each base variable that is merged by the ϕ -functions;
- then, allocate a single path-sensitive stack for all versions of the same base variable.

Optimized SSA forms



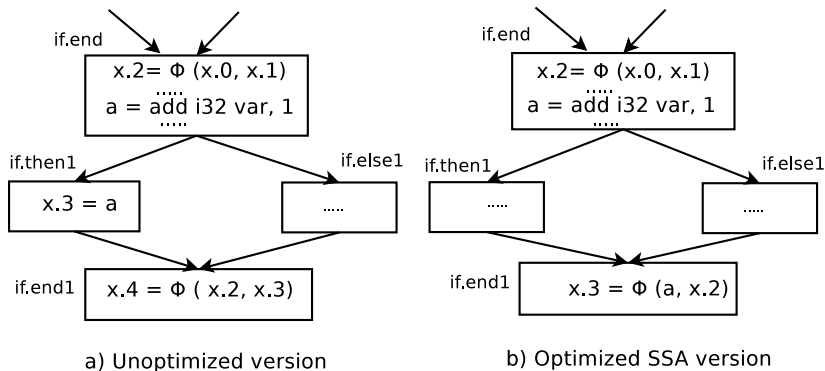
a) Unoptimized version



b) Optimized SSA version

a and x.2 are live simultaneously — hence, cannot be different versions of the same variable

Optimized SSA forms



a and x.2 are live simultaneously — hence, cannot be different versions of the same variable

in the above example, copy propagation breaks the *phi congruence property*...

ϕ – congruence property

Shreedhar et al. [SAS'99]

“The occurrences of all resources which belong to the same phi congruence class in a program can be replaced by a representative resource. After the replacement, the phi instruction can be eliminated without violating the semantics of the original program.”

- Sreedhar et al. circumvent the problem by translating the optimized SSA form to the conventional SSA form (that satisfies the phi congruence property) before translating out of SSA.
- **We directly build the HPSSA form over the optimized SSA form!**

Brief Algorithm

- **Insert τ -functions**
 - Insert at Thermal Frontiers
- **Allocate arguments to τ -functions**
 - path-sensitive traversal through the program to identify definitions that reach τ -functions through hot paths
 - constrains its inspection to only the ϕ -functions and the τ -functions

Allocating τ -function arguments

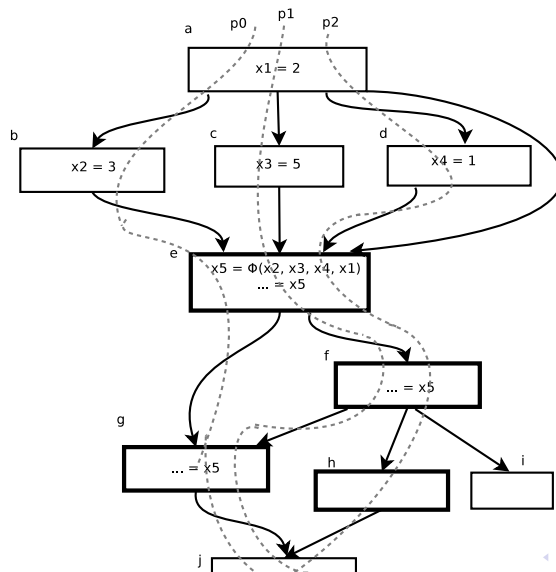
Uses a path-sensitive stack: **phiStack**

- **phiStack** is a stack of **frames**
- each frame $\langle d_i, \xi_i \rangle$ where $\xi_i = \{p_1, p_2, \dots\}$
- support operations:
 - push(frame f, block b)
 - pop(block b)

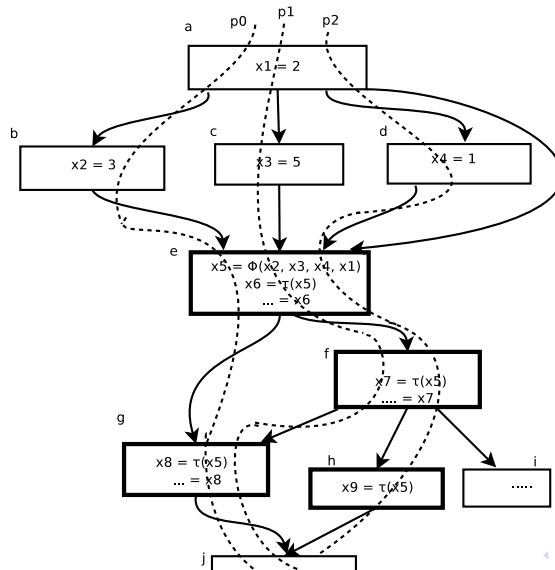
High-level algorithm

- Load arguments from ϕ - and τ -functions along with their hot path sets on **phiStack**
- Assign the definition from the topmost frame of **phiStack** to any τ -function encountered

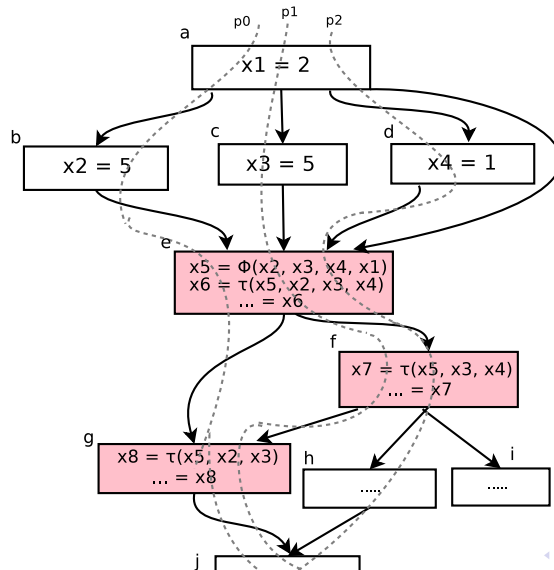
SSA program



τ -functions inserted



τ -arguments allocated



Outline

- 1 Introduction
- 2 The Hot Path SSA (HPSSA) Form
- 3 Constructing the HPSSA Form
- 4 Conclusions

Conclusions

- We propose a new algorithm that converts SSA programs to HPSSA. Being built over the SSA form, our new algorithm is much more suited for compiler frameworks that offer an SSA-based intermediate representation. This algorithm is also simpler and more efficient than the existing algorithm that builds HPSSA programs from non-SSA programs.
- Our algorithm is capable of operating on optimized SSA forms (resulting from compiler optimizations on the SSA form) that do not satisfy the *Phi Congruence Property*.
- We design an ILP-based path-profile guided register allocation. We provide experimental evaluation of the performance of our approach on eight benchmarks of SPEC CINT2000 benchmark suite.

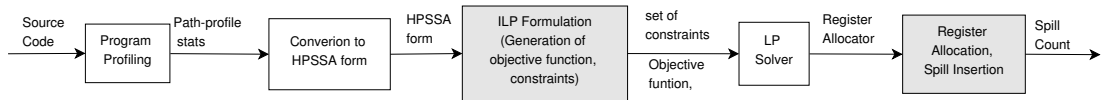
Questions

Questions?

dasd

- The occurrences of all resources which belong to the same phi congruence class in a program can be replaced by a representative resource. After the replacement, the phi instruction can be eliminated without violating the semantics of the original program.

dasd



dasd

```
1: Calculate all caloric connectors in the CFG of the program.
2: isInserted( $l, b$ ) = false for all instruction  $l$  and basic-block  $b$ 
3: Traverse CFG in topological order, performing the following:
4: for all  $\phi$ -function instruction  $l \in$  basic-block  $b$  do
5:   for all hot path  $p$  passing through  $b$  do
6:     for  $b_p =$  next basic-blocks on  $p$  till  $b_p \notin \text{DomFront}(b)$  do
7:       if  $b_p \in \text{CaloricConnectors} \wedge \neg \text{isInserted}(l, b_p)$  then
8:         insert a  $\tau$ -function  $x_{\text{new}} = \tau(x_{\phi_i})$  cor-
          responding to the  $\phi$ -function  $l$ :  $x_{\phi_i} =$ 
9:          $\phi(\dots)$  after all  $\phi$ -function statements
          in block  $b_p$ 
10:         $\text{isInserted}(l, b_p) = \text{true}$ 
11:      end if
12:    end for
13:  end for
```

dasd

While processing a basic block b ,

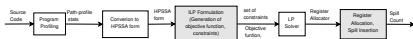
```

1: for all  $\phi$ -function  $l$ :  $x_o = \phi(x_1, x_2, \dots, x_n) \in b$  do
2:   for all  $x_i$  for which  $\Omega_{x_i}(b)$  exists do
3:      $\text{phiStack}_{x_i}.\text{push}(\Omega_{x_i}(b), b)$ 
4:   end for
5:   if  $b \in \text{IncubationNode}$  then
6:      $\text{frame} = []$ 
7:     for all  $(x_i, \xi_i) \in \text{phiStack}_{x_i}(b).Top$  do
8:        $\text{frame} = \text{frame.add}(x_i, \xi_i \cup \text{IncubPaths}(b))$ 
9:     end for
10:    for all arguments  $x_i$  in  $\phi$ -function  $l$  do
11:      if  $x_i$  is a concrete defn.  $\wedge b_p \rightarrow b$  an incoming edge then
12:         $\text{frame} = \text{frame.add}(x_i, \text{pathSet}(b_p \rightarrow b) \cup \text{IncubPaths}(b))$ 
13:      end if
14:      if  $x_i$  is a  $\phi$ -argument corresponding to a hot backedge then
15:         $\text{frame} = \text{frame.add}(x_i, \text{IncubPaths}(b))$ 
16:      end if
17:    end for
18:     $\text{phiStack}_x.\text{push}(\text{frame}, b)$ 
19:  end if
20: end for
21: for all  $\tau$ -function instructions  $l \in b$  do
22:   for all  $(x_i, \xi_i) \in \text{phiStack}_{x_i}(b).Top$  do
23:     if  $\xi_i \cap \text{pathSet}(b) \neq \emptyset$  then
24:        $\text{add}(x_i, \xi_i \cap \text{pathSet}(b))$  to speculative arguments of  $l$ 
25:     end if
26:   end for
27: end for
28: for all outgoing edge  $b \rightarrow b_s$  do
29:   if  $b_s$  is a join node then
30:     for all  $(x_i, \xi_i) \in \text{phiStack}_{x_i}.Top$  do
31:       if  $\xi_i \cap \text{pathSet}(b \rightarrow b_s) \neq \emptyset$  then
32:          $\Omega_{x_i}(b_s) = \xi_i \cap \text{pathSet}(b \rightarrow b_s)$ 
33:       end if
34:     end for
35:   end if
36: end for

```

37: Make a recursive call on the children of node b in the dominator tree, traversing them in topological of the nodes in the control-flow graph order.

dasd



dasd



dasd

