PROJECT 3

Function Inlining

Phan Nguyen

ECE-466

I. Implementation

1. Check if an Instruction is a CallIns:

Using isa<CallInst> to check whether it is a Call Instruction, then I will also check if the Called function get a basic block by calling has_Basicblock().

Inside the has_Basicblock() function, I get the Called Function and check if it is not null. If so, then I go and check if it has at least one basic block. Using the:

```
Function *fn = CallInst.getCalledFunction();

if (fn->begin() == fn->end())
```

If the above returns true, then it has no basic block inside it. If it has some basic block, then I will add the Call Instruction into a worklist and move to the next one.

2. Go through the work list:

Next I will iterate and check if the Called Function from the Call Instruction in work list is OK to inline by checking several things.

First, I need to check if the Called function size is within the limits by calling a function to loop through all of it instructions and return the size. This always needs to be done first, then if it is within the size; I will next check if the Const_arg flag is raised and if the Call Instruction has any constant argument at all using:

```
if (InlineConstArg && hasCont_Argu(*I));
```

Inside the hasCont_Argu is:

```
for(auto arg = I.arg_begin(); arg != I.arg_end(); ++arg){
    if (isa<Constant>(arg))
        return true;
```

I loop through the Call Instruction argument and check if any of them is a constant, if true then I increase the counter. Then after that I inline the function.

Every time I successfully inline a function, I will check the new size of that function by comparing it with the old size multiply the growth factor. If the size has grown over the limit then I will stop inlining.

II. Data Benchmark

Table for Instruction count:

Instruction:										
Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	643	334	239	239	239	334	334	334	419	239
arm	2007	801	390	801	668	732	409	721	784	373
basicmath	3129	522	313	522	406	522	522	522	591	313
bh	7716	2297	1869	2105	2159	2297	1931	1985	3301	1869
bitcount	665	423	423	423	423	423	423	423	665	423
crc32	236	112	83	112	112	112	112	112	145	83
dijkstra	867	484	216	303	303	484	484	484	322	216
em3d	2699	1078	634	870	776	1078	820	820	1233	615
fft	895	434	402	434	434	434	434	434	739	387
hanoi	96	51	51	51	51	51	51	51	96	51
hello	4	2	2	2	2	2	2	2	4	2
kmp	1175	568	338	568	384	568	370	568	559	324
l2lat	97	60	60	60	60	60	60	60	97	60
patricia	1826	681	472	522	472	681	681	681	1079	454
qsort	148	92	92	92	92	92	92	92	148	92
sha	3241	1621	375	607	607	1621	425	425	661	375
smatrix	409	247	198	247	198	247	247	247	315	198
sql	180866	104467	106798	106087	121768	104467	102720	104467	176711	102431
susan	27926	12998	6243	6829	6529	12998	7298	12998	12630	6243

Table for nInstruction Pre Inline:

Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	419	239	239	239	239	239	239	239	419	239
arm	784	373	373	373	373	373	373	373	784	373
basicmath	591	313	313	313	313	313	313	313	591	313
bh	3301	1869	1869	1869	1869	1869	1869	1869	3301	1869
bitcount	665	423	423	423	423	423	423	423	665	423
crc32	145	83	83	83	83	83	83	83	145	83
dijkstra	322	216	216	216	216	216	216	216	322	216
em3d	1233	615	615	615	615	615	615	615	1233	615
fft	739	387	387	387	387	387	387	387	739	387
hanoi	96	51	51	51	51	51	51	51	96	51
hello	4	2	2	2	2	2	2	2	4	2
kmp	559	324	324	324	324	324	324	324	559	324
l2lat	97	60	60	60	60	60	60	60	97	60
patricia	1079	454	454	454	454	454	454	454	1079	454
qsort	148	92	92	92	92	92	92	92	148	92
sha	661	375	375	375	375	375	375	375	661	375
smatrix	315	198	198	198	198	198	198	198	315	198
sql	176711	102431	102431	102431	102431	102431	102431	102431	176711	102431
susan	12630	6243	6243	6243	6243	6243	6243	6243	12630	6243

Table for nInstruction After Inline:

Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	643	334	239	239	239	334	334	334	419	239
arm	2007	836	390	836	694	758	409	747	784	373
basicmath	3129	655	313	655	412	655	655	655	591	313
bh	7716	2318	1869	2122	2173	2318	1933	1992	3301	1869
bitcount	665	423	423	423	423	423	423	423	665	423
crc32	236	116	83	116	116	116	116	116	145	83
dijkstra	867	484	216	303	303	484	484	484	322	216
em3d	2699	1085	634	875	781	1085	826	826	1233	615
fft	895	435	402	435	435	435	435	435	739	387
hanoi	96	51	51	51	51	51	51	51	96	51
hello	4	2	2	2	2	2	2	2	4	2
kmp	1175	574	338	574	384	574	370	574	559	324
l2lat	97	60	60	60	60	60	60	60	97	60
patricia	1826	697	478	529	478	697	697	697	1079	454
qsort	148	92	92	92	92	92	92	92	148	92
sha	3241	1660	375	618	618	1660	426	426	661	375
smatrix	409	247	198	247	198	247	247	247	315	198
sql	180866	104506	107302	106285	123169	104506	102729	104506	176711	102431
susan	27926	13255	6243	6858	6552	13255	7514	13255	12630	6243

Table for Inlined counter:

Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	1	1	0	0	0	1	1	1	(missing)	(missing)
arm	23	23	7	23	22	23	1	23	(missing)	(missing)
basicmath	13	13	0	13	3	13	13	13	(missing)	(missing)
bh	30	5	2	7	17	5	3	4	(missing)	(missing)
bitcount	0	0	0	0	0	0	0	0	(missing)	(missing)
crc32	1	1	0	1	1	1	1	1	(missing)	(missing)
dijkstra	5	5	1	4	4	5	5	5	(missing)	(missing)
em3d	20	20	8	18	16	20	12	12	(missing)	(missing)
fft	6	6	3	6	6	6	6	6	(missing)	(missing)
hanoi	0	0	0	0	0	0	0	0	(missing)	(missing)
hello	0	0	0	0	0	0	0	0	(missing)	(missing)
kmp	7	7	3	7	4	7	1	7	(missing)	(missing)
l2lat	0	0	0	0	0	0	0	0	(missing)	(missing)
patricia	16	10	8	9	8	10	10	10	(missing)	(missing)
qsort	0	0	0	0	0	0	0	0	(missing)	(missing)
sha	13	13	0	9	9	13	2	2	(missing)	(missing)
smatrix	1	1	0	1	0	1	1	1	(missing)	(missing)
sql	24	24	2199	209	1565	24	7	24	(missing)	(missing)
susan	26	26	0	15	11	26	11	26	(missing)	(missing)

Table for ConstArg counter:

Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	0	0	0	0	0	1	0	0	(missing)	(missing)
arm	0	0	0	0	0	3	0	0	(missing)	(missing)
basicmath	0	0	0	0	0	9	0	0	(missing)	(missing)
bh	0	0	0	0	0	3	0	0	(missing)	(missing)
bitcount	0	0	0	0	0	0	0	0	(missing)	(missing)
crc32	0	0	0	0	0	0	0	0	(missing)	(missing)
dijkstra	0	0	0	0	0	2	0	0	(missing)	(missing)
em3d	0	0	0	0	0	1	0	0	(missing)	(missing)
fft	0	0	0	0	0	3	0	0	(missing)	(missing)
hanoi	0	0	0	0	0	0	0	0	(missing)	(missing)
hello	0	0	0	0	0	0	0	0	(missing)	(missing)
kmp	0	0	0	0	0	1	0	0	(missing)	(missing)
l2lat	0	0	0	0	0	0	0	0	(missing)	(missing)
patricia	0	0	0	0	0	0	0	0	(missing)	(missing)
qsort	0	0	0	0	0	0	0	0	(missing)	(missing)
sha	0	0	0	0	0	3	0	0	(missing)	(missing)
smatrix	0	0	0	0	0	0	0	0	(missing)	(missing)
sql	0	0	0	0	0	9	0	0	(missing)	(missing)
susan	0	0	0	0	0	10	0	0	(missing)	(missing)

Table for SizerReq counter:

Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	1	1	1	1	1	1	0	0	(missing)	(missing)
arm	23	23	23	23	23	23	0	23	(missing)	(missing)
basicmath	13	13	13	13	13	13	13	13	(missing)	(missing)
bh	30	4	30	8	30	4	2	3	(missing)	(missing)
bitcount	0	0	0	0	0	0	0	0	(missing)	(missing)
crc32	1	1	1	1	1	1	0	1	(missing)	(missing)
dijkstra	5	5	5	5	5	5	4	4	(missing)	(missing)
em3d	20	20	20	20	20	20	11	11	(missing)	(missing)
fft	6	6	6	6	6	6	6	6	(missing)	(missing)
hanoi	0	0	0	0	0	0	0	0	(missing)	(missing)
hello	0	0	0	0	0	0	0	0	(missing)	(missing)
kmp	7	7	7	7	7	7	0	7	(missing)	(missing)
l2lat	0	0	0	0	0	0	0	0	(missing)	(missing)
patricia	16	10	10	10	10	10	10	10	(missing)	(missing)
qsort	0	0	0	0	0	0	0	0	(missing)	(missing)
sha	13	13	13	13	13	13	1	1	(missing)	(missing)
smatrix	1	1	1	1	1	1	1	1	(missing)	(missing)
sql	23	23	7574	229	2054	23	6	23	(missing)	(missing)
susan	26	26	26	26	26	26	10	26	(missing)	(missing)

Table for Timing execution:

Category	I	IO	IO10	IO100	IO50	IOA	IOG2	IOG4	NONE	0
adpcm	1.01	1.21	1.25	1.23	1.17	1.22	1.19	1.22	1.02	1.2
arm	0	0	0	0	0	0	0	0	0	0
basicmath	.02	.02	.03	.03	.01	.03	.03	.01	.02	.03
bh	.66	.81	.82	.81	.76	.8	.8	.79	.71	.8
bitcount	.12	.18	.18	.19	.18	.19	.18	.18	.13	.18
crc32	.11	.1	.09	.1	.09	.1	.09	.1	.11	.11
dijkstra	.03	.04	.04	.04	.04	.03	.04	.04	.03	.04
em3d	.24	.23	.25	.26	.26	.26	.26	.23	.28	.26
fft	.02	.02	.02	.02	.01	.02	.02	.01	.02	.02
hanoi	3.29	2.49	2.48	2.53	2.47	2.5	2.49	2.43	3.32	2.48
hello	0	0	0	0	0	0	0	0	(missing)	(missing)
kmp	.14	.13	.14	.14	.14	.14	.14	.12	.14	.13
l2lat	.01	.04	.04	.02	.03	.03	.05	.04	.02	.03
patricia	.03	.03	.03	.03	.03	.03	.03	.03	.03	.02
qsort	.02	.02	.02	.02	.02	.02	.02	.02	.02	.02
sha	.03	.03	.03	.03	.03	.03	.03	.03	.03	.03
smatrix	4.45	3.45	3.71	3.59	3.66	3.53	3.45	3.43	4.48	3.64
sql	0	0	0	0	0	0	0	0	0	0
susan	.45	.66	.97	.97	.98	.67	.67	.65	.44	.096

III. Data Analysis

1. Comparing number of Instruction after perfor inlining:

If we compare the number of Instruction after inlining in the adpcm, arm and basicmath for example. We see that the number of Instruction will increase greatly:

adpcm	from 419 to 643
arm	from 784 to 2007
basicmath	from 591 to 3129

When we extend the comparing to other tests, we see something similar in all cases. That is the instruction count always increases. And if we just leave it as is, the inling pass is not helping to optimize the program. Examples can be found within the timing graph; we can see that the program takes more time to complete compared to the pre-Inline version.

However, in some case inling does help the program to perform better if it is combined with other post-opt like mem2reg or others pass. When we compare the timing between "None" test, and "I0" test; we can always see the improvement in timing and decrease in number of Instructions.

But running an inling pass too aggressively is also not a good idea. Take the arm test for example:

It's inline counter for test that require the size of function within 10, 50 and 100 is different. While the "I010" test only has 390 instructions, the "I0100" has double it with 836 instructions. Comparing it to other tests, we see this trend also apply. This again means that the more we inline something, the more pressure we put on the program, since its instruction and size is increase.

In conclusion, Inlining pass is a good optimization pass to increase the performance of the program. But inlining too much will be a bad idea. From this project, we can see that if we pick the function that has number of instructions between 50 to 100 to perform inline. We also need to pay attention to the growth size of a function after inline, as if we leave it growth too big it will slow the program down. And last, inline can create more opportunities for other passes to optimize the program, so running a post-opt after inline pass is always a good idea.