Class exercise: Function inlining and specialization

COMP 40

November 21, 2011

Group

Keeper of the record:
Other group members:

Function inlining

In this problem I want you to *estimate the cost of function calls* by counting calls, returns, arithmetic operations, loads, compares, and branches.

What's to be gained by inlining `UArray_at(segment, r4)`? *Assume that after inlining, the compiler improves the code as much as possible.*

```
void *UArray_at(T array, int i) {
    assert(array);
    assert(i >= 0 && i < array->length);
    return array->elems + i*array->size;
}
```

	Calls & returns	*Arithmetic*	*Loads and stores*	*Comparisons*	*Branches*
Without inlining					
After inlining and specialization					

What's to be gained by inlining `Bitpack_getu(instr, 3, 6)`? *Assume that after inlining, the compiler improves the code as much as possible.*

```
static inline uint64_t shl(uint64_t word, unsigned bits) {
  assert(bits <= 64);
  if (bits == 64)
    return 0;
  else
    return word << bits;
}

static inline uint64_t shr(uint64_t word, unsigned bits) { // shift R logical
  assert(bits <= 64);
  if (bits == 64)
    return 0;
  else
    return word >> bits;
}

uint64_t Bitpack_getu(uint64_t word, unsigned width, unsigned lsb) {
  unsigned hi = lsb + width; // one beyond the most significant bit
  assert(hi <= 64);
  return shr(shl(word, 64 - hi), 64 - width); // different type of right shift
}
```

	Calls & returns	*Arithmetic*	*Loads and stores*	*Comparisons*	*Branches*
Without inlining					
After inlining and specialization					