

Dependency Analysis Using LLVM

//From runtime profile

```
#define edi_prof 100000
```

```
#define eax_prof 0
```

```
#define esi_prof 200000
```

```
#define ebx_prof 300000
```

```
#define ecx_prof 1000
```

```
void loop(int* mem)
```

```
{
```

```
    int edi = edi_prof;
```

```
    int eax = eax_prof;
```

```
    int esi = esi_prof;
```

```
    int ecx = ecx_prof;
```

```
    int ebx = ebx_prof;
```

```
    int edx;
```

```
    bb20:
```

```
        edx = *(mem+(edi + eax*4)/sizeof(int));
```

```
        edx += *(mem+(esi + eax*4)/sizeof(int));
```

```
        *(mem + (ebx+eax*4)/sizeof(int)) = edx;
```

```
        eax = eax+1;
```

```
        if(eax != ecx)
```

```
            goto bb20;
```

```
}
```

clang

BB

```
br label %BB1
```

LLVM IR

BB1

```
%eax.0 = phi i32 [ 0, %BB ], [ %13, %BB1 ]
```

```
%0 = shl nsw i32 %eax.0, 2
```

```
%1 = add nuw nsw i32 %0, 100000
```

```
%2 = lshr exact i32 %1, 2
```

```
%3 = getelementptr inbounds i32* %mem, i32 %2
```

```
%4 = load i32* %3, align 4, !tbaa !1
```

```
%5 = add nuw nsw i32 %0, 200000
```

```
%6 = lshr exact i32 %5, 2
```

```
%7 = getelementptr inbounds i32* %mem, i32 %6
```

```
%8 = load i32* %7, align 4, !tbaa !1
```

```
%9 = add nsw i32 %8, %4
```

```
%10 = add nuw nsw i32 %0, 300000
```

```
%11 = lshr exact i32 %10, 2
```

```
%12 = getelementptr inbounds i32* %mem, i32 %11
```

```
store i32 %9, i32* %12, align 4, !tbaa !1
```

```
%13 = add nuw nsw i32 %eax.0, 1
```

```
%14 = icmp eq i32 %13, 1000
```

```
br i1 %14, label %BB2, label %BB1
```

BB2

```
ret void
```

Preprocessing +
profile info

Banerjee's
Method



No alias between
the two loads
and the store

Output from Disassembler

```
...
20: mov  (%edi,%eax,4),%edx
23: add  (%esi,%eax,4),%edx
26: mov  %edx,(%ebx,%eax,4)
29: add  $0x1,%eax
2c: cmp  %ecx,%eax
2e: jne  20
```

```
void loop(int* restrict memLd1, int* restrict memLd2, int* restrict memSt,
          int edi, int eax, int esi, int ecx, int ebx)
```

```
{
    int edx;
```

```
    bb20:
```

```
        edx = *(memLd1+(edi + eax*4)/sizeof(int));
```

```
        edx += *(memLd2+ (esi + eax*4)/sizeof(int));
```

```
        *(memSt + (ebx+eax*4)/sizeof(int)) = edx;
```

```
        eax = eax+1;
```

```
        if(eax != ecx)
```

```
            goto bb20;
```

```
}
```

Separate memory ports
for non-aliasing
accesses, facilitating
datapath synthesis

Parallelizable loop

Input to Accelerator Synthesis