Scheduling Weakly Consistent C Programs for Reconfigurable Hardware

Nadesh Ramanathan, John Wickerson, George Constantinides.

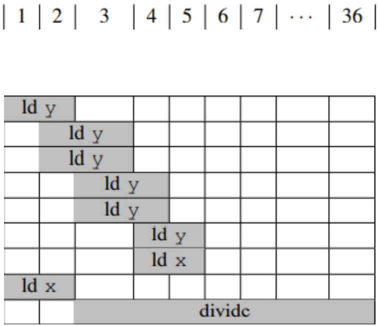
Presented by
Akshay Gopalakrishnan

February 17, 2022

Who are they?

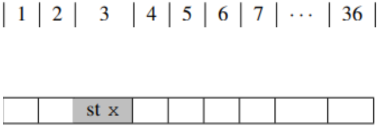
Example

```
int r0=0,r1=0,r2=0;
r0=y+y+y+y+y+y;
r1=x;
r2=x/a;
```



Example

```
x=1;
```

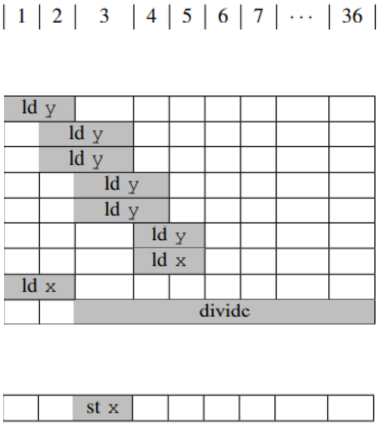


Example

```
int r0=0,r1=0,r2=0;
r0=y+y+y+y+y+y;
r1=x;
r2=x/a;
```

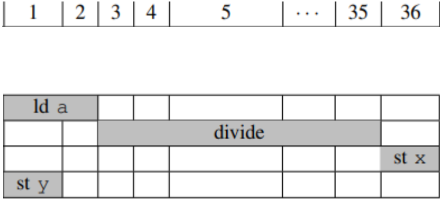
```
x=1;
```

assert($r1 = 1 \Rightarrow r2 \neq 0$)



Example


```
x=a/3;
y=1;
```



Example

```
int r0=0,r1=0;
r0=y;
if(r0==1) r1=x;
```

1	2	3	4	5	...	35	36
		ld y					
		ld x					
				slt y==1? x:null			

Example


```
x=a/3;
y=1;
```

```
int r0=0,r1=0;
r0=y;
if(r0==1) r1=x;
```

$\mathtt{assert}(\mathtt{r0} = 1 \Rightarrow \mathtt{r1} = 1)$

1	2	3	4	5	...	35	36
ld a							
		divide					
							st x
st y							

		ld y					
		ld x					
				slt y==1? x:null			

Problem of Dependencies

$$E_{\text{intra-iter}} = \{(v, v', 0) \mid sb(v, v') \wedge sloc(v, v') \wedge (v \in V_{\text{st}} \vee v' \in V_{\text{st}})\}$$

$$E_{\text{inter-iter}} = \{(v, v', 1) \mid sloc(v, v') \wedge (v \in V_{\text{st}} \vee v' \in V_{\text{st}})\}.$$

Adding WW—WR—RW—RR Dependancies

$$E_{\mathrm{sc}\downarrow} = \{(v, v', 0) \mid sb(v, v') \wedge v \in V_{\mathrm{sc}}\}$$
$$E_{\mathrm{sc}\uparrow} = \{(v, v', 0) \mid sb(v, v') \wedge v' \in V_{\mathrm{sc}}\}$$

```
int r0=0,r1=0,r2=0;
r0=y+y+y+y+y+y;
r1=x;
r2=x/a;
```

```
x=a/3;
y=1;
```

Final Dependency Expression

$$E^{\text{pipe}}_{\text{mem,SC}} = E_{\text{intra-iter}} \cup E_{\text{inter-iter}} \cup E_{\text{at}\downarrow} \cup E_{\text{at}\uparrow} \cup E_{\text{at-inter-iter}}$$

Example

Without pipelining

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12
`r1=x;`	ld_{na} x						ld_{na} x					
`r2=ld(&y,ACQ);`			ld_{ACQ} y						ld_{ACQ} y			
`r0=z;`					ld_{na} z						ld_{na} z	

With pipelining

Cycle:	1	2	3	4	5	6	7	8	9	10
`r1=x;`	ld_{na} x				ld_{na} x					
`r2=ld(&y,ACQ);`			ld_{ACQ} y				ld_{ACQ} y			
`r0=z;`					ld_{na} z				ld_{na} z	

Weakening: Adding Release-Acquire Dependencies

$$E_{\mathrm{acq}\downarrow} = \{(v, v', 0) \mid sb(v, v') \wedge v \in V_{\mathrm{acq}}\}$$
$$E_{\mathrm{rel}\uparrow} = \{(v, v', 0) \mid sb(v, v') \wedge v' \in V_{\mathrm{rel}}\}$$

```
x=a/3;
y=1;
```

```
int r0=0,r1=0;
r0=y;
if(r0==1) r1=x;
```

Adding RR Dependency

$$E_{\mathrm{RAR}} = \{(v, v', 0) \mid sb(v, v') \wedge sloc(v, v') \wedge v \in V_{\mathrm{at}} \cap V_{\mathrm{ld}} \wedge v' \in V_{\mathrm{at}} \cap V_{\mathrm{ld}}\}.$$

```
int r0=0,r1=0,r2=0;
r0=y+y+y+y+y+y;
r1=x;
r2=x/a;
```

Final Dependency Expression

$$
\begin{aligned}
E^{\text{pipe}}_{\text{mem,weak}} = {} & E_{\text{intra-iter}} \cup E_{\text{inter-iter}} \cup E_{\text{sc}\downarrow} \cup E_{\text{sc}\uparrow} \cup \\
& E_{\text{acq}\downarrow} \cup E_{\text{rel}\uparrow} \cup E_{\text{RAR}} \cup \\
& E_{\text{sc-inter-iter}} \cup E_{\text{acq-inter-iter}} \cup \\
& E_{\text{rel-inter-iter}} \cup E_{\text{RAR-inter-iter}}
\end{aligned}
$$

.

Example

Without pipelining

Cycle:	1	2	3	4	5	6	7	8
`r1=x;`	ld_{na} x				ld_{na} x			
`r2=ld(&y,ACQ);`	ld_{ACQ} y				ld_{ACQ} y			
`r3=z;`			ld_{na} z				ld_{na} z	

With pipelining

Cycle:	1	2	3	4	5	6
`r1=x;`	ld_{na} x		ld_{na} x			
`r2=ld(&y,ACQ);`	ld_{ACQ} y		ld_{ACQ} y			
`r3=z;`			ld_{na} z		ld_{na} z	

Testing on Message Passing

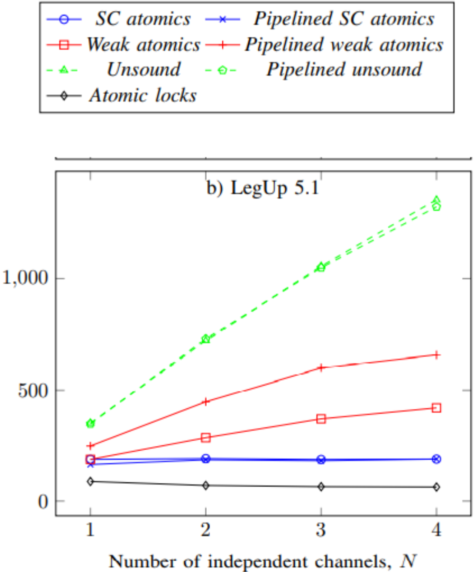
```
atomic_int flag1 = 0, ..., flagN = 0;
int data1 = 0, ..., dataN = 0;
```

```
for(i=0; i<ITER; i++) {
  if(ld(&flag1,ACQ)==0){
   data1++;
   st(&flag1,1,REL);
  }
  ...
  if(ld(&flagN,ACQ)==0){
   dataN++;
   st(&flagN,1,REL);
  }
}
```

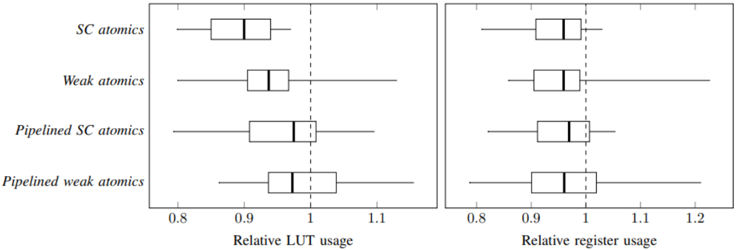
```
for(i=0; i<ITER; i++) {
  if(ld(&flag1,ACQ)==1){
   data1++;
   st(&flag1,0,REL);
  }
  ...
  if(ld(&flagN,ACQ)==1){
   dataN++;
   st(&flagN,0,REL);
  }
}
```

Fig. 5. A two-threaded message-passing example with *acquire-release* semantics on N independent channels.

Testing on Message Passing



Pros and Cons



Conclusion