How to Write Fast Numerical Code

Spring 2011
Lecture 11

Instructor: Markus Püschel
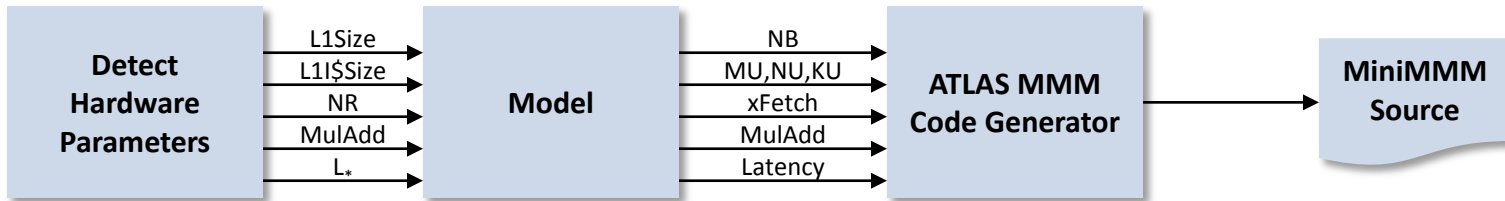
TA: Georg Ofenbeck

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Miscellaneous

- **Intel compiler icc:**
  - Free for linux
  - Who needs it for Visual Studio?

Last Time: Model-Based ATLAS

```
┌──────────────┐  L1Size  ┌──────────┐   NB      ┌──────────────┐        ┌──────────┐
│    Detect    │─────────▶│          │──────────▶│              │        │ MiniMMM  │
│   Hardware   │ L1I$Size │          │ MU,NU,KU  │  ATLAS MMM   │───────▶│  Source  │
│  Parameters  │─────────▶│  Model   │──────────▶│Code Generator│        │          │
│              │   NR     │          │  xFetch   │              │        └──────────┘
│              │─────────▶│          │──────────▶│              │
│              │ MulAdd   │          │  MulAdd   │              │
│              │─────────▶│          │──────────▶│              │
│              │   L*     │          │  Latency  │              │
│              │─────────▶│          │──────────▶│              │
└──────────────┘          └──────────┘           └──────────────┘
```

- **Search for parameters replaced by model to compute them**
- **More hardware parameters needed**

Today: Remaining Details

- Register renaming and the refined model for x86

- TLB effects

Dependencies

- **Read-after-write (RAW) or true dependency**

W $\quad r_1 = r_3 + r_4$ $\qquad$ *nothing can be done*
R $\quad r_2 = 2r_1$ $\qquad\qquad$ *no ILP*

- **Write after read (WAR) or antidependency**

R $\quad r_1 = r_2 + r_3$ $\qquad$ *dependency only by* $\qquad$ $r_1 = r_2 + r_3$ $\qquad$ *now ILP*
W $\quad r_2 = r_4 + r_5$ $\qquad$ *name → rename* $\qquad\qquad$ $r\ \ = r_4 + r_5$

- **Write after write (WAW) or output dependency**

W $\quad r_1 = r_2 + r_3$ $\qquad$ *dependency only by* $\qquad$ $r_1 = r_2 + r_3$
$\quad$ … $\qquad\qquad\qquad$ *name → rename* $\qquad\qquad$ … $\qquad\qquad$ *now ILP*
W $\quad r_1 = r_4 + r_5$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $r\ \ = r_4 + r_5$

Resolving WAR

R $r_1 = r_2 + r_3$ *dependency only by* $r_1 = r_2 + r_3$ *now ILP*
W $r_2 = r_4 + r_5$ *name → rename* $r\ \ = r_4 + r_5$

- **Compiler: Use a different register, r = r_6**

- **Hardware (if supported): register renaming**
  - Requires a separation of architectural and physical registers
  - Requires more physical than architectural registers

Register Renaming

physical registers   *ISA*   architectural (logical) registers

				r1
				r2
				r3
				rn
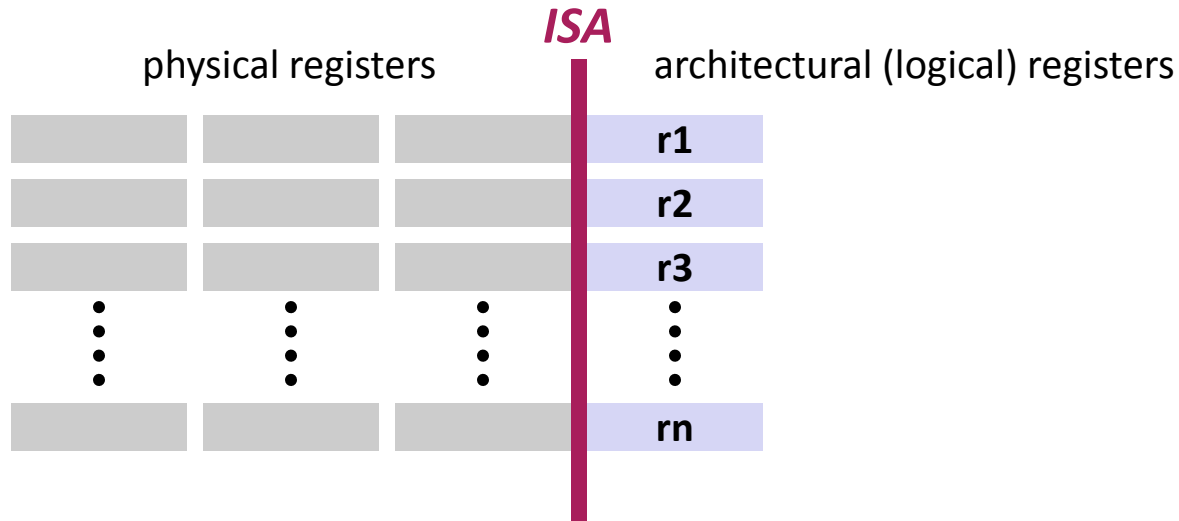
- **Hardware manages mapping architectural → physical registers**

- **More physical than logical registers**

- **Hence: more instances of each r_i can be created**

- **Used in superscalar architectures (e.g., Intel Core) to increase ILP by resolving WAR dependencies**

Scalar Replacement Again

- **How to avoid WAR and WAW in your basic block source code**

- **Solution: Single static assignment (SSA) code:**
  - Each variable is assigned exactly once

no duplicates

```
<more>
s266 = (t287 - t285);
s267 = (t282 + t286);
s268 = (t282 - t286);
s269 = (t284 + t288);
s270 = (t284 - t288);
s271 = (0.5*(t271 + t280));
s272 = (0.5*(t271 - t280));
s273 = (0.5*((t281 + t283) - (t285 + t287)));
s274 = (0.5*(s265 - s266));
t289 = ((9.0*s272) + (5.4*s273));
t290 = ((5.4*s272) + (12.6*s273));
t291 = ((1.8*s271) + (1.2*s274));
t292 = ((1.2*s271) + (2.4*s274));
a122 = (1.8*(t269 - t278));
a123 = (1.8*s267);
a124 = (1.8*s269);
t293 = ((a122 - a123) + a124);
a125 = (1.8*(t267 - t276));
t294 = (a125 + a123 + a124);
t295 = ((a125 - a122) + (3.6*s267));
t296 = (a122 + a125 + (3.6*s269));
<more>
```

Micro-MMM Standard Model

- MU*NU + MU + NU ≤ NR − ceil((Lx+1)/2)

- Core: MU = 2, NU = 3

reuse in a, b, c

- Code sketch (KU = 1)

```
rc1 = c[0,0], …, rc6 = c[1,2] // 6 registers
loop over k {
  load a  // 2 registers
  load b  // 3 registers
  compute // 6 indep. mults, 6 indep. adds, reuse a and b
}
c[0,0] = rc1, …, c[1,2] = rc6
```

Extended Model (x86)

- **MU = 1, NU = NR – 2 = 14**

a   b   =   c   *reuse in c*

- **Code sketch (KU = 1)**

```
rc1 = c[0], …, rc14 = c[13] // 14 registers
loop over k {
  load a            // 1 register
  rb  = b[1]        // 1 register
  rb  = rb*a        // mult (two-operand)
  rc1 = rc1 + rb    // add  (two-operand)
  rb = b[2]         // reuse register (WAR: renaming resolves it)
  rb = rb*a
  rc2 = rc2 + rb

  …
}
c[0] = rc1, …, c[13] = rc14
```

Summary:
- **no reuse in a and b**
+ **larger tile size for c**

Today: Remaining Details

- **Register renaming and the refined model for x86**

- **TLB effects**
  - Blackboard

Experiments

- *Unleashed:* **Not generated = hand-written contributed code**

- *Refined model* **for computing register tiles on x86**

- **Blocking is for L1 cache**

- *Result:* **Model-based is comparable to search-based (except Itanium)**

graph: Pingali, Yotov, Cornell U.