18-643 Lecture 7: C-to-HW Synthesis:

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today: develop a mental model for how to turn “proper” C into “proper” HW, whether by a compiler or by hand

• Notices
  – Handout #4: lab 1, due noon, 9/27
  – Project status report due each Friday

• Readings (see lecture schedule online)
  – for textbook treatment: Ch 7, Reconfigurable Computing
C as Model of Computation for HW?

• Common arguments for using C to design HW
  – easy algorithm specification
  – popularity, popularity, popularity

• A large semantic gap to bridge
  – sequential thread of control
  – abstract time
  – abstract I/O model
  – functions only have a cost when executing
  – missing structural notions: bit width, ports, modules

• No problem getting HW from C, but good HW?

All sequential, imperative languages
A Program is a Functional-Level Spec

int fibr(int n) {
    if (n==0) return 0;
    if (n==1) return 1;

    return fibr(n-1)+fibr(n-2);
}
A Program is a Functional-Level Spec

```c
int fibm(int n) {
    int *array,*ptr; int i;

    if (n==0) return 0;
    if (n==1) return 1;

    array=malloc(sizeof(int)*(n+1));
    array[0]=0; array[1]=1;

    for(i=2,ptr=array ; i<=n ; i++,ptr++)
        *(ptr+2)=*(ptr+1)+*ptr;

    i=array[n];
    free(array);
    return i;
}
```
A Program is a Functional-Level Spec

```c
int fibi(int n) {
    int last=1; int lastlast=0; int temp;

    if (n==0) return 0;
    if (n==1) return 1;

    for(;n>1;n--) {
        temp=last+lastlast;
        lastlast=last;
        last=temp;
    }

    return temp;
}
```
Opening Questions

• Do they all compute the same “function”?

• Should they all lead to the same hardware?

• Should they all lead to “good” hardware?
  – what does recursion look like in hardware?
  – what does `malloc` look like in hardware?
What is in a C Function?

• What it specifies?
  – abstracted data types (e.g., int, floats, doubles)
  – step-by-step procedure to compute the return value from input arguments
  – a sequentialized execution

• What it doesn’t specify?
  – encoding of the variables
  – where the state variables are stored
  – what types and how many functional units to use
  – execution timing, neither in terms of wall-clock time, clock cycles, or instruction count
  – what is strictly necessary for correctness
Mapping Program to Hardware

• For you to produce “good” structural RTL
  – identify suitable “temporal and spatial pattern”
  – flesh out concrete datapath (bit/cycle exact)
  – develop correct and efficient control sequencing

• C-to-HW (i.e., C-to-RTL) compiler bridges the gap between functionality and implementation
  – extract parallelism from a sequential specification
  – fill in the details below the functional abstraction
  – make good decisions when filling in the details

*Keep in mind: what you don’t need to specify you also can’t control*
A Look at Scheduling and Allocation
Procedural Block to Data Flow Graph

\[
\begin{cases}
  x = b; \\
  \text{if} \ (y) \\
  \quad x = x + a;
\end{cases}
\]

\[
\begin{cases}
  x_1 = b; \\
  \text{if} \ (y) \\
  \quad x_2 = x_1 + a; \\
  \text{else} \\
  \quad x_2 = x_1 \\
  x = x_2
\end{cases}
\]

static elaboration to single-assignment
Data Flow Graph

- Captures data dependence irrespective of program order
  - nodes=operator
  - edge=data flow
- “Work” is total delay if done sequentially
  - e.g., if delay(+)\(=1\), delay(*)\(=2\), work = 6
- “Critical path” is the longest path from input to output
  - e.g., critical path delay = 4
  - no implementation can complete faster than critical path delay

\[
\begin{align*}
v &= a + b; \\
w &= b * c; \\
x &= v + c; \\
y &= v + w; \\
z &= x + y;
\end{align*}
\]

\[z\] only care about
Program-Order, Sequential Mapping

- Need only one of each functional unit type: 1 adder, 1 multiplier
- Delay equal “work”: 6

In contrast, if combinational
- 4 adder, 1 multiplier
- delay=4

Is there a shorter schedule for
1 adder and 1 multiplier?
Optimized Sequential Mapping

- In general,
  - given a set of functional units, what is the shortest schedule
  - given a schedule, what is the minimum set of functional units
  - given a target delay (>= critical path), find a min-cost schedule

- Very efficient algorithms exist for solving the above

- Harder part is setting the right goal
  - minimum delay could be expensive
  - minimum resource could be slow

```
delay=4 using 1 adder and 1 multiplier
```
Generating Datapath

How do I know 3 registers are needed?
Control FSM

- Assume initially $a$ in $r_1$; $b$ in $r_2$; $c$ in $r_3$

<table>
<thead>
<tr>
<th></th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>add</th>
<th>mult</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel1</td>
<td>en1</td>
<td>sel2</td>
<td>en2</td>
<td>sel3</td>
<td>en3</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>add</td>
<td>1</td>
<td>mul</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>add</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
It should remind you of this

inputs

FSM

datapath

outputs

clock
Good Hardware Needs Concurrency
Where to Find Parallelism in C?

- C-program has a sequential reading
- Scheduling exploits operation-level parallelism in a basic block (≈ work/critical-path-delay)
  - “ILP” is dependent on scope
  - techniques exist to enlarge basic blocks and to increase operation-level parallelisms: loop-unrolling, loop pipelining, superblock, trace scheduling, etc.

  Many ideas first developed for VLIW compilation

- Structured parallelism can be found across loop iterations, e.g., data parallel loops
Loop Unrolling

\[
\begin{align*}
\text{for}(i=0; i<N; i++) & \quad \text{for}(i=0; i<N; i+=2) \\
\quad v &= a[i]+b[i]; & \quad v &= a[i]+b[i] \\
\quad w &= b[i]*c[i]; & \quad w &= b[i]*c[i] \\
\quad x &= v+c[i]; & \quad x &= v+c[i] \\
\quad y &= v+w; & \quad y &= v+w \\
\quad z[i] &= x+y; & \quad z[i] &= x+y; \\
\quad v_0 &= a[i+1]+b[i+1]; & \quad v_0 &= a[i+1]+b[i+1] \\
\quad w_0 &= b[i+1]*c[i+1]; & \quad w_0 &= b[i+1]*c[i+1] \\
\quad x_0 &= v_0+c[i+1]; & \quad x_0 &= v_0+c[i+1] \\
\quad y_0 &= v_0+w_0; & \quad y_0 &= v_0+w_0 \\
\quad z[i+1] &= x_0+y_0; & \quad z[i+1] &= x_0+y_0 \\
\end{align*}
\]

\textit{data-parallel iterations}

work=?? critical path=??
Loop Pipelining

\begin{align*}
&\text{for}(i=0; i<N; i++) \quad \text{for}(i=1; i<N; i++) \quad \text{for}(i=2; i<N; i++) \\
&\{ \quad \{ \quad \{ \\
&\quad v = a[i]+b[i]; \quad v' = v; \quad v = a[i]+b[i]; \quad v' = v; \quad v = a[i]+b[i]; \\
&\quad w = b[i]*c[i]; \quad w' = w; \quad w = b[i]*c[i]; \quad w' = w; \quad w = b[i]*c[i]; \\
&\quad x = v+c[i]; \quad x = v'+c[i-1]; \quad x = v'+c[i-1]; \quad x = v'+c[i-1]; \quad x = v'+c[i-1]; \\
&\quad y = v+w; \quad y = v'+w'; \quad y = v'+w'; \quad y = v'+w'; \quad y = v'+w'; \\
&\quad z[i] = x+y; \quad z[i-1] = x+y; \quad z[i-1] = x+y; \quad z[i-1] = x+y; \quad z[i-1] = x+y; \\
&\} \quad \} \quad \} \\
&x = v+c[i-1]; \\
&y = v+w; \\
&z[i-1] = x+y; \\
&z[i-2] = x'+y'; \\
&z[i-2] = x+y; \\
&x = v'+c[i-1]; \\
&y = v'+w'; \\
&z[i-1] = x+y; \\
&\}
\end{align*}
Pipelined Loop

for(i=2;i<N;i++) {
    v' = v; w' = w;
    x' = x; y' = y;
    v = a[i]+b[i];
    w = b[i]*c[i];
    x = v'+c[i-1];
    y = v'+w';
    z[i-2] = x'+y';
}

- In SW, loop pipelining increases producer-consumer distance
- In HW, work on parts of 3 different iterations in same cycle

work=?? critical path=??
Pipelined Loop

```c
for(i=2;i<N;i++) {
    v' = v; w' = w;
    x' = x; y' = y;
    v = a[i]+b[i];
    w = b[i]*c[i];
    x = v'+c[i-1];
    y = v'+w';
    z[i-2]= x'+y';
}
```

- In SW, loop pipelining increases producer-consumer distance
- In HW, work on parts of 3 different iterations in same cycle

This looks more familiar?
How Hard is MMM?

```c
float A[N][N], B[N][N], C[N][N];

for(int i=0; i<N; i++) {
    for(int j=0; j<N; j++) {
        for(int k=0; k<N; k++) {
            C[i][j]=C[i][j]+A[i][k]*B[k][j];
        }
    }
}
```

- $T_1$, $T_\infty$, $P_{avg}=T_1/T_\infty$?
- # of memory access?
- $T_1$/ # of memory access?
Compute Throughput Needs Data Throughput
A Look at dependency & memory access

\[
\text{for}(i=0; \ i<N; \ i++) \\
\text{for}(j=0; \ j<N; \ j++) \\
\text{for}(k=0; \ k<N; \ k++) \\
C[i][j] += A[i][k]*B[k][j]
\]

(1) Assume row-major layout and large 2-power \( N \)
(2) 64-Byte DRAM interface and 8-KByte row buffer
Loop Reordering

```c
for(k=0; k<N; k++)
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            C[i][j] += A[i][k]*B[k][j]
```

Data-parallel over the i and j loops
**Code Structure to HW Concurrency**

What about strided b access?

\[
C[i][j] += f(i, j, 0) + f(i, j, 1) + \ldots + f(i, j, k-1)
\]

\[O(N^3)\] memory access necessary?

\[
C[i][j] += f(i, j, k)
\]

pipelined j-loop

unrolled k and pipeline j
Memory not Monolithic Abstraction

- Control memory organization to match access pattern

![Diagram showing control memory organization with banks and data connections]
Control over Data Layout

- An array of N words; index is $\lg_2 N$ bits
  \[ \text{Lg}_2 N \text{ array index (sequential)} \]

- N-word total storage
  - divided into B banks; bank number is $\lg_2 B$ bits
  - each bank is W-word wide; word-select is $\lg_2 W$ bits
  - line index within bank is $\lg_2 (N/B/W)$ bits

- Assign bank #, word select and index to maximize
  - spatial locality in word select
  - “entropy” in bank #

In general interleaved & reordered
Example: Image Frame

• N pixels in $\sqrt{N}$-by-$\sqrt{N}$ frame
• Spatial locality in $\sqrt{W}$-by-$\sqrt{W}$ tiles
• Parallelism across same-column tiles

Can you tell the compiler (through C) this is what you want?
Parting Thoughts

• C-to-HW compiler fills in details between algorithm and implementation

• No magic—good HW only if it is in the program
  – not every computation is right for HW so not every C-program is right for HW
  – even for right ones, how the C is written matters

• C-to-HW technology is very real today
  – work very well on some domain or applications
  – has blindspots; need human-in-the-loop pragmas

  Useful in different ways to an expert HW designer vs. a so-so HW designer vs. a SW programmer