Lecture #8: Performance
or How I Learned to Stop Worrying
and Love the Parallelism

Paul Hartke
Phartke@stanford.edu
Stanford EE183
April 29, 2002

Lab Stuff

• Lab #1 writeup due **TODAY** at midnight
  – Keep considering how to create an all
electronic version of the writeup
• General Lab #2 Questions?
Performance

• In many applications, performance is the name of the game
• As long as Moore’s law continues to hold, performance is gained by waiting and doing nothing…
  – We (the logic design community) benefit directly from the efforts of the Silicon process technology folks
  – Much more important long term trend than architectural improvement
  – That means that a change in Moore’s Law would have a huge impact

Parallelism is the Key

• However, we get paid/graded for doing logic design… 😊
• The main arrow in our quiver is parallelism
  – Do more work in a given unit of wall clock time
• Two methods
  – Replication
    • Add more units
  – Pipelining
    • Use the existing units more efficiently
Replication

• Instead of having only one unit calculate each fractal position, bisect the screen and have two units calculating fractal values
• Linear Speedup does not often happen
  – Does using two units improve the performance by two for an application? *Rarely*
  – Law of Diminishing Returns kicks in earlier than usually expected

Losses through Impedance Mismatches

• The algorithm cannot usually be perfectly partitioned
  – Leverage Symmetry
    • Partition horizontally or vertically for fractal?
• Conflicts arise from shared resource accesses
  – Single Write Port in design so it must be shared
    • Solve the problem by delaying the access of one unit
      – That is, the unit is stalled until the resource become free
      – You *never* get that time back!!!
  – What is the duty cycle of each unit’s access?
    • So not an issue for the fractals
Cycle Time Tradeoffs

- Increased cycle time allows more logic to be performed in a given clock cycle
  - Remember there are setup and hold constraints for edge triggered (DFF) based designs
- Longer cycle time means more stuff is computed before the control logic has a chance to react to the datapath
- Really moot since you can rarely pick your cycle time
  - Often picked to be the “sweet-spot” of the Silicon process and cell library

Fractal Generation Datapath

![Fractal Generation Datapath Diagram]

- Feedback for next iteration
- PipeRegisters
Multi-Cycle Paths

• Why did we insert registers?
• Why not just let the datapath take N clock cycles?
• Just say no!!!!
  – Risk versus Reward
  – Tool chains do not support this
    • Same arguments really as register retiming
      – Is this just inertia? I personally don’t think so…

Pipelining??

• Inserting registers did not decrease the wall clock time?!?!?!
  – Simply made our design fully synchronous
  – Was not pipelining
• What is the utilization of the various stages?
Path of Data

• Insert the next data item into the datapath before the previous one has finished
• PipeRegisters keep the computation separate
• Increases utilization for operators
• What is the effect of the algorithm feeding back on itself?
  – Do all iterations have the same number of iterations?
    • How to manage this?
Paradigm

- If you have N stages in your pipeline, think of it as N units overlapped in space and offset in time
  - What if one of the N virtual units need to communicate to another?
  - “Forward” the data from one to another
- Pipeline Performance Losses
  - The datapath rarely can be evenly split
    - Depends on the longest path in a stage
  - PipeRegisters add fixed overhead (setup and hold)
  - Structural “Hazards” can cause stalls/bubbles in the pipeline

What performance is required?

- Replication and Pipling are not trivial to implement—make sure you need them
- Is either needed for Lab #2?
- How would you tell?
  - Hint: each Julia image takes
    \[(64*64*64*7*1/50e6) = 0.036s\] to create.
  - Is this “real-time” enough for an animation?
Architectural Changes **Do Matter**

- Eliminating Work is *always* Good
- Architectural changes that eliminate work are always good  
  - fixed point versus floating point
- Moving work to another part of the design that is less timing critical is also important

Aside: **SuperLinear Speedup**

- SuperLinear Speedup is possible when the problem is structured so that work is actually *avoided*
  - Caches  
    - Memory access avoidance
  - Binary Search  
    - Once answer value is found then signal the other units to stop
- However, make sure you count *all* the costs in your cost benefit analysis.
How to Pipeline for Lab #2?

- Create the datapath in a single module
  - Use the coregen multiplier
  - Register all inputs and outputs
  - Implement and then run the static timing tool
  - Add a pipe stage
  - Iterate
- Note that the coregen multiplier has several pipeline options

Static Timing Tool

- Must fully implement design first
  - Needs placement and routing info
- Tools ➔ Simulation/Verification ➔ Interactive Timing Analyzer
- Analyze ➔ Against Auto Generated Design Constraints
  - Why does just using the Auto constraints work?
- Most Critical Paths will be shown
  - You can crossprobe into the Floorplanner
Fixed Point Thoughts?

• Any more thoughts on the fixed point sizes of the datapath?
  – Want to minimize these since multipliers grow quickly in size and latency with operand size
  – Don’t want them so small that overflow of the integer part occurs (results in aliasing) or that the fractional part has large quantization error
• We stop the loop when magnitude is greater than 4
  – Use that knowledge to approximate size of intermediate operands
• Matlab’s Simulink Fixed Point Block Set