<table>
<thead>
<tr>
<th>Load Value Speculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Value Prediction</td>
</tr>
<tr>
<td>- Value Locality</td>
</tr>
<tr>
<td>- Speculative Execution</td>
</tr>
<tr>
<td>- Instruction Level Parallelism</td>
</tr>
<tr>
<td>- Superscalar Processors</td>
</tr>
</tbody>
</table>
Needs

- ILP through branch prediction limited due to data dependences
- Even infinite sources and perfect prediction wouldn't help more
- Need to eliminate data serialization
Background

- All instructions actually predictable!
- But loads are most predictable and incur longest latencies.
- Hence better to predict only loads.
On the cache side

- Even one cache miss per one hundred accesses can double a program’s execution time
- Over half the runtime is spent stalling for loads that miss in the second-level cache
Plausible?

- 32 bit word => $2^{32}$ values
- 64 bit word => $2^{64}$ values
- How to speculate?
- But fortunately exhibit *Value Locality*
Basics

- Predict an instruction result and forward to its dependent instructions
- If correct, no problem
- O/W, need recovery mechanism
Figure 1. Pipeline Stages of Hardware Value Speculation Mechanism for Flow Dependent Instructions. The dependent instruction executes with the predicted value in the same cycle as the predicted instruction.
Details

- Cycle-penalty for mispredictions too costly
- At times not enough information for accurate prediction
- Supply *Confidence Estimators to avoid predictions unlikely to be correct*
Basic/Last Value predictor

- Predict last executed value
- Can only predict sequences like 3, 3, 3, 3, 3, 3....
- But luckily they are quite frequent
Types - II

- **Stride 2-delta predictor (ST2D)**
  - Remembers last value for each load
  - Also, maintains a stride
  - Inhibits two consecutive mispredictions
- **Last 4 Value predictor (L4V)**
  - 4 most recently used values
  - Predicts alternating sequences
    -1, 0, -1, 0, -1, 0, -1, 0...
  - and sequences no longer than 4 values
Finite Context Method predictor (FCM)

- Uses hashing
- Predicts any load using the same sequence
- In addition to the capabilities of the aforementioned predictors
Types - V

- **Differential Finite Context Method (DFCM)**
  - Improves FCM
  - Predicts values never seen before
  - Adds complexity
Comparison of Prediction Accuracy

Figure 1. Performance of predictors when all loads are predicted
Conclusion

- Power dissipation
- Energy consumption
- Complex hardware
- Good prediction
- Accuracy
- Latency
Conclusions

- Increasing the predictor size increases the performance only as long as the access latency remains reasonably low.

- 100% accurate branch predictor with a two-cycle latency performs worse than a relatively inaccurate branch predictor with single-cycle latency.
Conclusions

- best predictors in the literature, actually perform well only on loads that hit in the cache

- For loads that miss, these more complex predictors are no better than the much simpler ones

- In other words, for the loads that need speculation the most, the simpler, smaller, and faster predictors perform as well as the more complex predictors.
Final Word

- Power dissipation & energy consumption first-order design constraint
  => complex hardware is unappealing.

- can expect good performance from seemingly simple predictors.