# MIPS-I Integer Processor Simulator augmented with a Load Value Predictor

Anand Singh Kunwar
Computer Science and Engineering
Indian Institute of Technology, Kanpur
Roll No: 13110
Email: anandk@iitk.ac.in

R Sundararajan
Computer Science and Engineering
Indian Institute of Technology, Kanpur
Roll No: 13523
Email: rsundar@iitk.ac.in

***Abstract*—We extend the MIPS-I Integer Processor Simulator by converting the single cycle MEM stage to a multi-cycle MEM stage. We explore the notion of value locality, a different locality in comparison to temporal and spatial. By inculcating load value predictor(s) in our modified MIPS-I Simulator we predict 32-bit register values using previously seen values. We shall try to show any performance gains from this inculcation.**

***Keywords*—*Load Value Predictor, Value Locality, MIPS Simulator***

## I. Introduction

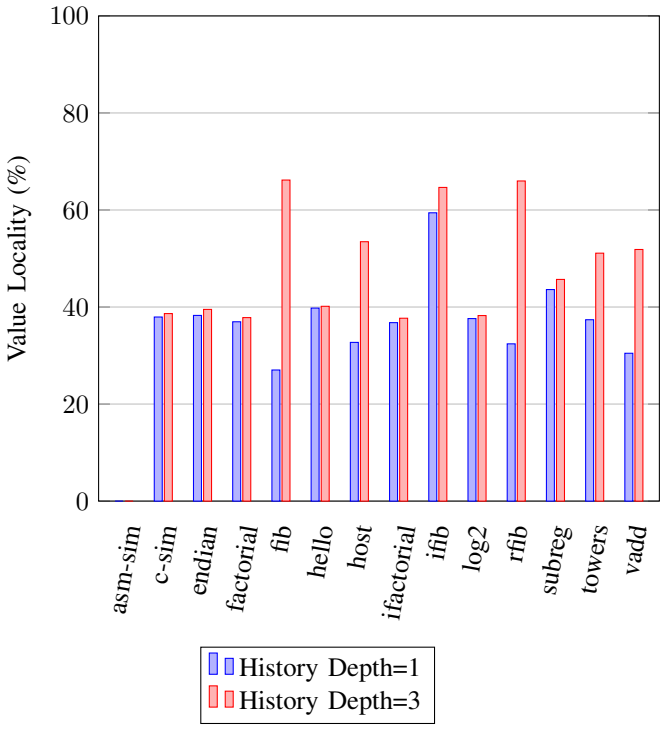
Processor clock speeds are much higher than speed for memory fetching. To combat the latency of the memory accesses, spatial and temporal locality is exploited by caches for performance gains. In this project, we use value locality[1], a construct we shall be using in our Load Value Predictor(LVP). We'll include LVP(s) in our MIPS-I Simulator and try to show its performance benefits. We'll slightly modify our MIPS-I pipeline to have a multi-stage MEM.

## II. Value Locality

In the paper by Mikko H. Lipasti et al.[1], value locality is define as "the likelihood of a previously-seen value recurring repeatedly within a storage location". We try to examining the value locality of general purpose registers immediately after the loads from memory. Unlike branch prediction, where we need to predict a single bit, here we try to predict 32-bit values. We do this by looking at the history that we associate with the program counter and use a function(say, mode) that takes the history of that hash and makes the prediction. In the adjoining bar chart, we depict the value locality percentages for history depth as 1 and history depth as 3 for all the benchmarks. The value locality for each benchmark is measured by counting the number of times each static load instruction retrieves a value from memory that matches a previously-seen value for that static load and dividing by the total number of dynamic loads in the benchmark.

## III. Load Value Predictor

Using the construct of Value Locality, we create a Load Value Predictor. This predictor can reduce the latency due to loads by multiple cycles in our model of modified MIPS that has a multi-cycle MEM stage described later. The Load Value




Predictor has two major components the Load Value Prediction(LVP) Table and Load Classification Table(LCT). We have skipped the Constant Verification Unit(CVU) described in the original paper by Mikko H. Lipasti et al for simplicity.[1]. We have various LVP Unit configurations namely Simple, Constant and Limit and can be seen in Table I.

TABLE I. LVP Unit Configurations

| LVP Unit Configuration | LVP Table | | LCT | |
|---|---|---|---|---|
| | Entries | History Depth | Entries | Bits / Entry |
| Simple | 1024 | 1 | 32 | 2 |
| Constant | 1024 | 1 | 128 | 1 |
| Limit | 4096 | 16 | 128 | 2 |

### A. Load Value Prediction(LVP) Table

We have a LVP Table which has some entries. Each entry of the table corresponds to a hash of the program counter(of the load instruction). Each entry contains a history of last seen

32-bit values corresponding to that program counter. We vary the history size for different configurations of the LVP Unit.

### B. Load Classification Table(LCT)

This is like a meta predictor that tells us the processor whether to use the prediction performed using the LVP Table. We have a saturating $n$-bit counter with varying the number of entries for different configurations of the LVP Unit.

## IV. MIPS-I Simulator Extension

### A. Making the MEM stage multi-cycled

For better measuring of CPI improvements after the addition of the load value predictor, we made the MEM stage multicycled. Say the total number of MEM stages is N. For the first (N-1) MEM stages, all the corresponding MEM stage does is to copy the value from the pipeline registers preceeding it to the pipeline registers succeeding it. All the actual work happens in the $N^{th}$ MEM stage. Hence it can be observed that in case of a load instruction, the value will be available only at the end of the Nth MEM stage. The code that has been written is for a generic pipeline with N MEM stages and we have tested the 14 benchmarks for N=$\{1, 2, 3$ and $4\}$. This pipeline consists of bypass paths from MEM(i) to EX for i=$\{1$ to N-1$\}$. When we encounter a load instruction followed by an instruction (any of the N+1 instructions followin it) depending on it, we stall the dependent instruction until the value from the load becomes available. It is exactly these stalls that we would be looking to reduce with our load value predictor. In Table II, we showcase CPIs for benchmarks without the LVP and just interlocks for data hazards wherein a value from a load is needed before it is computed.

TABLE II. Base Benchmarks for MIPS-I(Modified)

| Benchmarks | CPI varying number of cycles in MEM stage | | | |
|---|---|---|---|---|
| | 1 Cycle(s) | 2 Cycle(s) | 3 Cycle(s) | 4 Cycle(s) |
| asm-sim | 1.151 | 1.208 | 1.269 | 1.358 |
| c-sim | 1.008 | 1.203 | 1.310 | 1.418 |
| endian | 1.007 | 1.195 | 1.301 | 1.411 |
| factorial | 1.006 | 1.185 | 1.283 | 1.382 |
| fib | 1.000 | 1.133 | 1.201 | 1.270 |
| hello | 1.009 | 1.226 | 1.344 | 1.462 |
| host | 1.005 | 1.190 | 1.293 | 1.402 |
| ifactorial | 1.006 | 1.186 | 1.284 | 1.383 |
| ifib | 1.002 | 1.155 | 1.240 | 1.331 |
| log2 | 1.007 | 1.205 | 1.314 | 1.425 |
| rfib | 1.001 | 1.139 | 1.212 | 1.285 |
| subreg | 1.008 | 1.220 | 1.337 | 1.456 |
| towers | 1.002 | 1.186 | 1.287 | 1.395 |
| vadd | 1.002 | 1.072 | 1.109 | 1.146 |

### B. Augmenting the pipeline with the Load Value Predictor (LVP)

For a load instruction, the LVP generates the prediction (from LVPT) and whether to use the prediction (from LCT) in the second half of the ID stage. It is in the ID stage that we detect that the particular instruction is dependent on a previous load and whether we can use the prediction for that particular load. Hence, we can't generate the predictions for a load instructions after the ID stage because we might possibly need the values in the ID stage (in the same cycle). For eg. if a load is followed by an instruction dependent on it, then if we generate the predictions in the EX stage of the load then we can't simultaneously have the values in the ID stage. This design therefore needs bypass paths from ID to ID, EX to ID and MEM(i) to ID (i=$\{1$ to N-1$\}$) to pass the predicted value (if any) to the ID stage. After augmenting the MIPS simulator with the LVP, we have tested for N=$\{2, 3\}$ on the benchmarks. In Table III, we showcase the improvements in CPIs over the baseline (without LVP) for the benchmarks for the 2-MEM and 3-MEM stage cases.

TABLE III. Benchmarks for MIPS-I (modified)

| Benchmarks | MIPS-I (2 MEM Cycle(s)) | | | |
|---|---|---|---|---|
| | Base | Simple | Constant | Limit |
| asm-sim | 1.208 | 1.208 | 1.208 | 1.208 |
| c-sim | 1.203 | 1.100 | 1.097 | 1.100 |
| endian | 1.195 | 1.096 | 1.095 | 1.096 |
| factorial | 1.185 | 1.104 | 1.097 | 1.104 |
| fib | 1.133 | 1.129 | 1.147 | 1.130 |
| hello | 1.226 | 1.111 | 1.109 | 1.111 |
| host | 1.190 | 1.164 | 1.167 | 1.165 |
| ifactorial | 1.186 | 1.105 | 1.099 | 1.105 |
| ifib | 1.155 | 1.089 | 1.085 | 1.089 |
| log2 | 1.205 | 1.113 | 1.104 | 1.113 |
| rfib | 1.139 | 1.126 | 1.141 | 1.126 |
| subreg | 1.220 | 1.133 | 1.131 | 1.135 |
| towers | 1.186 | 1.155 | 1.164 | 1.151 |
| vadd | 1.072 | 1.045 | 1.045 | 1.045 |
| Benchmarks | MIPS-I (3 MEM Cycle(s)) | | | |
| | Base | Simple | Constant | Limit |
| asm-sim | 1.269 | 1.269 | 1.269 | 1.269 |
| c-sim | 1.310 | 1.156 | 1.150 | 1.156 |
| endian | 1.301 | 1.153 | 1.149 | 1.153 |
| factorial | 1.283 | 1.167 | 1.158 | 1.167 |
| hello | 1.344 | 1.172 | 1.167 | 1.172 |
| host | 1.293 | 1.255 | 1.259 | 1.261 |
| ifactorial | 1.284 | 1.162 | 1.153 | 1.162 |
| ifib | 1.240 | 1.141 | 1.135 | 1.142 |
| log2 | 1.314 | 1.175 | 1.162 | 1.175 |
| rfib | 1.212 | 1.270 | 1.273 | 1.271 |
| subreg | 1.337 | 1.207 | 1.202 | 1.210 |
| towers | 1.287 | 1.242 | 1.251 | 1.237 |
| vadd | 1.109 | 1.069 | 1.068 | 1.069 |

## V. Inferences from Table III

For the 2-MEM cycle case, we can observe improvements in the CPI (compared to the base CPI) for all benchmarks except asm-sim and subreg. Similarly, for the 3-MEM cycle case, we can improvements for all benchmarks except rfib and asm-sim. We can observe that the asm-sim benchmark does gives the same CPI both with and without the LVP. This is because, it consists of no data hazards wherein the value of a load is needed before it is computed. Also, not much of a difference can be observed in CPIs with the Simple, Constant and Limit configurations of the LVP. This is because none of the programs are tailor made to be most optimally improved by either of these configurations.

## VI. Conclusion

We observe that some programs have a significant boost in CPI due to the Load Value Predictor we included in our MIPS Integer Processor Simulator. We observe that these programs generally have high value locality percentages and we can conclude that value locality has a correlation with improvement in CPI. Further extensions of this project include stress testing using programs that exploit value locality, modelling the perfect load value predictor that always predicts the correct value if the program has seen it before and observing CPI changes on that LVP.

## References

[1] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," *ACM SIGPLAN Notices*, vol. 31, no. 9, pp. 138–147, 1996.