# Value-Prediction-Based Data Trace Compression in BERI

Allison Pearce
St. Edmund's College
ap819@cam.ac.uk

*Abstract*—**Extended program traces are a vital tool for the development and analysis of research processors, but they are often restrictively large. In the case of BERI, a software debugging unit streams trace entries that are transmitted from the processor over a JTAG serial interface. The low throughput of JTAG makes trace compression particularly valuable. This paper describes a hardware implementation of data trace compression for BERI based on the VPC algorithms for value-prediction-based trace compression in software. Some elements of VPC proved difficult to translate into hardware or were not adaptable to a streaming application, but the model of using predictors to compress trace data at the hardware level is promising. This implementation achieves a 4.6 compression ratio, which is lower than the best academic research implementations, but is on par with industry standards.**

## I. Introduction

Traces are a common and often necessary tool for debugging and analyzing processors and software. Address traces track the PC during the execution of a program. Extended traces or data traces provide additional information, such as register content and branch tags. For example, Cambridge's Bluespec Extensible RISC Implementation (BERI) [21] generates a 256-bit trace entry for each instruction. Needless to say, full data traces can be extremely large, and storing them is a problem. In many systems, including BERI, traces are sent from the processor over a JTAG interface. JTAG's low throughput makes transmission of large amounts of trace data difficult.

Compression helps to alleviate the storage and transmission problems. Compression can be implemented in hardware or software. Software solutions usually fall into one of two categories: regeneration and compression. Execution-driven solutions like ATOM [5] create binaries that regenerate a new trace on demand, eliminating the need to store the trace at all. However, trace regeneration can be too slow or too expensive for some uses. It is also ISA specific, difficult to port, and can produce unrepeatable results for nondeterministic programs. Compression solves the issues of nondeterminism, but storing compressed traces still requires some disk space. Compressed traces must also be decompressed before they can be used.

Prediction-based compression strategies have been extensively researched. The value-prediction-based compression (VPC) algorithms attempt to minimize the drawbacks of compression and were expressly designed for use with extended traces [3]. Value predictors recognize patterns and extrapolate them to predict the values most likely to occur next. Value predictors have been shown to successfully predict the content of CPU registers using fast, simple algorithms [4] [6] [7] [16] [19]. CPU registers tend to contain data similar to that found in data traces. When compressing traces with VPC, value predictors forecast the most likely values for the next PC and extended trace data. If one of the predictors is successful, only the identifier of that value predictor need be written, instead of the full trace entry. Zcompr uses a combination of prediction and interpretation to produce even better compression than VPC [12].

VPC and Zcompr, like other software compression strategies, focus on alleviating the problem of *storing* traces. This does not help with the real-time transmission of the trace, and it requires an extra step after the trace has been produced. Hardware compression allows for quick transmission in addition to the reduced storage overhead, and compressing the trace as it is generated means that no additional steps are required after running the processor.

The majority of hardware trace compression techniques focus on address traces. One popular address compression technique is to simply discard contiguous addresses and only report jumps or branches [2] [11] [9]. The Nexus 5001 Forum defined a standard for packet-based tracing using a similar address filtering strategy [1]. These strategies have low overhead and are easy to implement. However, they are limited by the size of the average basic block, which is usually only four or five instructions [15]. Another strategy is to output the difference between each successive address, as in the MIPS PDTrace module [10]. With this method, compression is limited because consecutive instructions cannot be skipped. All of these techniques achieve compression ratios between 5 and 10x [14]. Research implementations have achieved significantly better compression results. Kao et al. proposed an LZ-compression-based mechanism that achieves up to 454:1 compression ratios, but at a high complexity (50,000 gates) [13]. Uzelac et al. developed a technique using prediction structures that they claim delivers a 1098:1 compression ratio [20].

There has been comparatively little research on hardware compression for extended trace data. One system, RATCHET [18], achieves 20-30:1 compression using a filter cache. However, RATCHET's traces do not capture branching behavior, so they are only reliable for cache behavior analysis, not program analysis. Industry standard compression ratios for for hardware extended trace compression are also on the order of 5-10:1 [8].

I implemented an extended trace compression mechanism in hardware for BERI based on VPC. The primary goal of compression was to reduce transmission time of traces being streamed by a software debugging application in real time. Because of the inherent differences between hardware and software and in streaming versus storage, VPC could not be perfectly mapped to BERI, but many of the techniques and ideas could. This project demonstrates the feasibility of porting VPC-like prediction algorithms to hardware to achieve compression rates on par with industry standards.

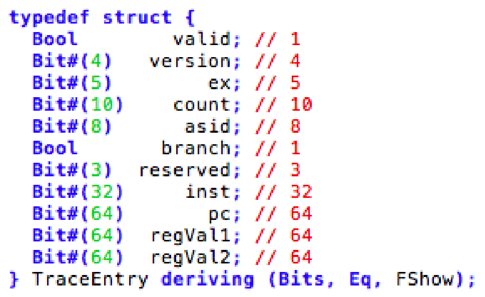
## II. Background

This project's hardware trace compression mechanism was based on the VPC algorithms. VPC is single-pass, which is convenient for storage because it means that the trace need never be fully uncompressed, and it is necessary for compressing traces for transmission during real time streaming. In VPC1, 37 value predictors are applied to the trace data. In the event of a successful prediction, the predictor identification code is written to a file after being further compressed to one byte by a dynamic Huffman encoder. If there are no successful predictions, a flag is written to the file, followed by the unpredictable data. The output of VPC1 is quite compressible, so VPC2 adds a stage to do so. VPC3 uses fewer value predictors and attempts to separately predict both the PC and the trace data. In VPC3, the compressed trace output is divided into four streams that can be further compressed easily: identifiers of PC predictors, unpredictable PC data, identifiers of trace data predictors, and unpredictable trace data. Each of the four streams is then compressed with BZIP2. VPC4 is autogenerated from VPC3 and is faster and better compressing.

In this project, I used some of the same value predictors as VPC3, including the finite context method (FCMn) [17]. FCMn hashes the $n$ most recently encountered values to serve as an index into a history table. The value at this index is used as the prediction, hoping that the value that follows in this case will be the same as the value that followed last time this series of $n$ values was encountered. The value at the index is updated with the current value (PC or trace data) if the prediction was wrong. FCMn predictors are useful for long, arbitrary repeating sequences such as PCs.

Last $n$ value predictors (LnV) [4] are another history-based predictor. They simply use the $n$ most recent values as $n$ predictions. LnV predictors perform well with values that alternate or with repeating patterns that have a period of no more than $n$.

Stride 2-delta predictors (ST2D) [6] store the most recent value and the difference (delta) between the most recent value and the one that preceded it. The predicted value is the most recent value plus the difference. It also maintains a second delta value for a second prediction. The second delta is only updated when the same difference is encountered twice in a row. ST2D predictors perform well at predicting numbers in sequences that are incremented or decremented by a fixed amount, such as registers containing loop indices and other types of counters. ST2D predictors were not used in VPC3, but they were found to be advantageous in the BERI compressor.



```
typedef struct {
  Bool        valid; // 1
  Bit#(4)   version; // 4
  Bit#(5)        ex; // 5
  Bit#(10)    count; // 10
  Bit#(8)      asid; // 8
  Bool       branch; // 1
  Bit#(3)  reserved; // 3
  Bit#(32)     inst; // 32
  Bit#(64)       pc; // 64
  Bit#(64)  regVal1; // 64
  Bit#(64)  regVal2; // 64
} TraceEntry deriving (Bits, Eq, FShow);
```

Fig. 1. BERI's 256-bit TraceEntry struct

## III. Compression Implementation

The compression mechanism described here is in some ways a combination of VPC1 and VPC3. Like VPC3, we use a smaller subset of the most successful predictors to forecast both the PC and the trace data. Because the primary use case requires traces to be individually compressed and quickly decompressed for real-time streaming, the strategy of writing to multiple streams and performing BZIP2 compression on the streams is not applicable. Like VPC1, all data is written to the same output. If the predictions are incorrect, a flag and the unpredictable data are transmitted. The predictors used are a subset of those implemented in VPC3. PCs are predicted with FCM1ab (finite context method using the single most recent PC and providing two predictions) and an FCM3ab (finite context method using the three most recent PCs and providing two predictions) for a total of four predictions. Trace data is predicted using L4V (last $n$ value predictor with $n = 4$) and stride 2-delta algorithms (six total predictions).

PC predictions were necessarily implemented with global index information, meaning that the PC predictions are based on the most recent PC values in the whole program's execution. The trace predictions are PC-specific, so the L4V predictor returns the last four trace values encountered at a particular PC, and the ST2D predictor returns the most recent value and delta for the previous occurrence of that PC. If multiple predictions are correct for a particular field, ties are broken by selecting the one with the highest use count.

Unlike any of the VPC algorithms, I divided the BERI trace data into groups of fields and predicted each group separately. This is because some of the trace fields are much easier to predict than others, and some respond better to different prediction techniques. For example, BERI traces include an increasing `count` field, which is predicted reasonably well using ST2D but not at all with LnV. If the compressor is designed to treat all trace data as a unit, then correctly predicting all the data but one register, or correctly predicting all of the data but not all with the same predictor, still means that the whole prediction is invalid and the full trace must be transmitted. Predicting groups of fields separately from the rest of the data allows for some compression even when the more difficult predictions fail. The optimal data groups were experimentally determined.

The VPC software algorithms base their predictors on one- or two-level hash tables implemented using C arrays. The hardware implementation relied on BRAM modules to implement hash tables. I did not use any two-level tables in the final hardware compression implementation, because there was no significant benefit to prediction accuracy to offset the additional delay required to index into both levels of the table for lookups and updates.

The compression module interfaces with BERI's DebugUnit. A secondary goal of the project was to change the existing structure of BERI as little as possible. The compressor provides a GetPut interface where the DebugUnit can enqueue trace entries for compression. A combination of methods and rules performs the lookups and updates with the BRAM predictor tables. The results are packed into a `CompressedTraceEntry`, a struct containing a valid bit, the four-bit version, three-bit predictor identifiers for the PC and the groups of trace data, and some combination of mispredicted data or padding to fill 256 bits. The DebugUnit expects a 256-bit value, but based on the prediction results encoded

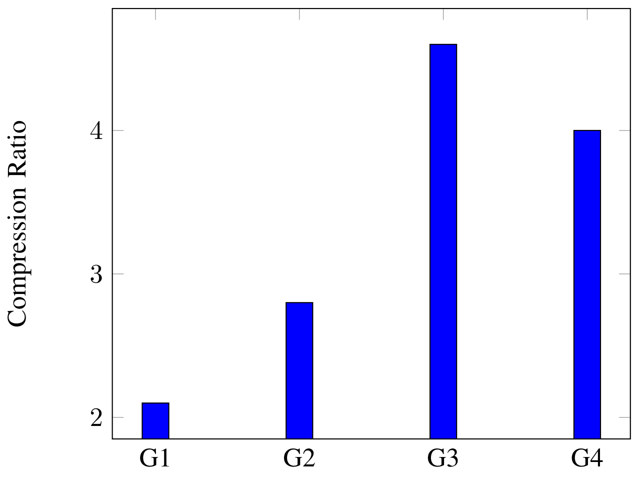


Fig. 2. Compression ratios for different prediction groupings of trace fields. All groups predicted ex, asid, branch, reserved, and inst as one group and predicted PC individually. G1 predicted regVal1, regVal2, and count as one unit of data. G2 used regVal1 as a group and regVal2 with count as a second group. G3 predicted count, regVal1, and regVal2 separately. G4 predicted the registers separately and transmitted the count value uncompressed.

in the version field, it will send only the first $n$ bytes in a packet to the software stream tracing application. If all fields are predicted correctly, only three bytes are needed (for the valid bit, version number, and predictor identifiers). If a unit of trace data is mispredicted, the correct value is sent with the message packet. Exceptions occur if the "easy data"[1] is mispredicted or if all of the other data fields are mispredicted; in this case, the original trace entry is sent uncompressed. This is in part to reduce the complication of the decompression logic and in part because some of the sixteen possible versions are reserved, so the number of states that can be communicated to the DebugUnit and decompresser is more limited.

## IV. Evaluation Methods

### A. System

Experiments were performed running a simulation of the BERI processor on an Intel Core i5 running Ubuntu 3.2.0 with four 1197MHz CPUs, 32kB L1 caches, 256kB L2 cache, and 8192kB L3 cache.

### B. Timing Measurements

As in the analysis of the VPC algorithms, runtime was measured using the UNIX shell `time` command. The sum of the user and system time is reported, which equates to reporting the CPU time.

### C. Traces

Compression was testing by tracing a BERI simulation running a short program for 268,084 instructions. The program included a variety of operations with implementations of bubble sort, quick sort, and modular exponentiation. It is a weakness of this experiment that only one program was available to run on the simulated BERI processor.

## V. Results

### A. Prediction Groups

The fields in a BERI trace are show in Figure 1. The count and register fields change in the most complex manner, and we experimented to find the optimal number of predictors and groups of data to maximize compression. Figure 2 shows the compression results for the four best performing groupings. The valid and version fields are necessary for the trace transmission and are not compressed; they contribute 5 bits. In the optimal division, the PC is predicted and compressed individually. The `ex, asid, branch, reserved` and `inst` fields almost never change for a given PC and are easy to predict, so they are predicted as one group. Each of the `count, regVal1,` and `regVal2` fields are predicted separately. Not shown in the figure are the results attained with the same divisions as the optimal grouping but with the `count` predictor indexed globally instead of with the PC. That is to say, the ST2D predictor that was used to forecast the `count` field stored the deltas between the most recent count values, regardless of PC. Using this strategy resulted in extremely poor compression, indicating that `count` is closely tied to the PC and PC-related data.

It is also important to note that we had a limited sample of programs to experiment with. These data groups were optimal for the data available, but it is possible that others would perform better when considering a wider variety of program samples.

### B. Compression Rate

The best performing division of trace data (PC, each register, and count predicted individually, other fields predicted as a unit) produced a compression ratio of 4.6, reducing the volume of data transmitted from the processor by 78%. Figure 3 compares this compression ratio to some other popular software and hardware compression algorithms. The BERI implementation is on par with the compression mechanisms used in industry, but it is outperformed by research implementations of hardware and especially software data trace compression.

### C. Compression Time

Running the simulation and test program with no compression took an average of 27.61 milliseconds. With hardware compression, the simulation took 27.71 milliseconds, an increase of only 0.37%. Modifying the software debugging application to perform the decompression was out of the scope of this project, so unfortunately no data is available on transmission or decompression time.

## VI. Discussion

The BERI compression module could certainly be improved in many ways, but it achieves a compression rate comparable to industry standard. Moreover, RATCHET, the better-performing hardware data trace compressor, has major limitations: namely, its inability to capture branching behavior. Software data tracing has been studied more extensively, but many of the enhancements that made VCP3 and VPC4 so fast and effective are either infeasible or impractical for hardware stream tracing. Still, we can see that the use of value prediction techniques for compression is sound, and prediction is already widely implemented and highly effective in hardware such as branch predictors.

A major divergence in design from VPC was the decision to predict some fields of the trace data separately instead of as a single unit. This demonstrates a tradeoff between making more accurate predictions and maximizing compression. There are two ways to increase prediction accuracy: use more predictors, or predict more fields individually so as to avoid the complex

[1]The groups of data are further explained in the Results section

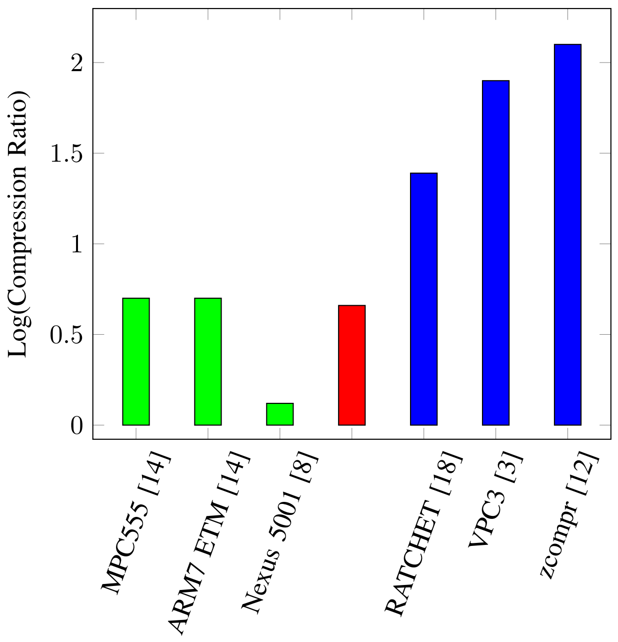


Fig. 3. Comparison of log of compression rate for a sample of industry and academic implementations. The blue bars indicate industry standard, green bars indicate academic research, and the red bar is for the compressor in this paper. Note that zcompr and VPC3 are software compressors; all others are hardware

dynamics of multiple fields changing in different ways. But both of these strategies increase the minimum number of bits needed for predictor identifiers (either in the form of more bits per predictor encoding or more predictor encodings). This increases the minimum size of the transmission when all data is predicted perfectly and thus decreases the maximum compression attainable.

We experimented with finding optimal values for many parameters, but there are almost limitless opportunities to continue tweaking values in hopes of perfecting the predictors. The number of bits in the keys for the BRAMs and the prioritization of the predictors are just two of the values that affect the outcomes of the predictors. Further analysis of these parameters could yield better results. Similarly, the minimum trace size for each entry in this implementation is three bytes. With more experiments and fine-tuning, it would be possible to reduce this number. But any strategy in which each entry is compressed in isolation has a theoretical maximum compression rate equal to the size of the original data to be transmitted (256 in this case, if it were possible to compress each 256-bit trace to one bit of data). VPC3 and VPC4 achieve significantly better results by further compressing the streams with BZIP2, but BZIP2 does not yet have a hardware implementation. Other strategies predict multiple values at once, or use consistent patterns to avoid sending data until the pattern changes. Exploring some form of one of these strategies has the potential to yield far better returns. To have a truly effective compression algorithm competitive with the best research implementations, one needs to be able to send less than one bit of data per trace entry.

## VII. Future Work

The relatively short time frame for this project left many things unimplemented or unexplored. There are two obvious and necessary next steps: alter the software stream tracing application to decompress the traces, and performing synthesis and hardware cost analysis of the compressor module. Other avenues for investigation include finding a hardware analogue to BZIP2 in VPC3 that could be used to further compress the outputs of the predictors and experimenting with other value predictors. It may also be possible to encode multiple trace entries at once by predicting the data for a sequence of instructions. This would require more substantial changes to the hardware DebugUnit and the debugging software, but it would allow for much greater compression. Finally, this compression implementation is specific to the current BERI TraceEntry. Changes to the TraceEntry content could significantly alter the performance of the predictor. It would be ideal to have a more flexible compressor that could handle a variety of trace entry types with different data exhibiting different patterns.

## VIII. Conclusion

I implemented a value-predictor-based trace compression mechanism in hardware that delivers a 4.6:1 compression ratio. This ratio is on par with many industry hardware compression implementations. Recent academic approaches achieve compression ratios that are better, but the best of the academic examples are based on similar techniques of prediction. This project was extremely valuable for learning about compression and prediction techniques, learning Bluespec, and understanding more of the fundamental design differences between software programming languages and hardware development languages.

## References

[1] IEEE-ISTO 5001-1999, the Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface. 1999.

[2] ARM. ARM embedded trace macrocell architecture specification. 2007.

[3] Martin Burtscher, Ilya Ganusov, Sandra J Jackson, Jian Ke, Paruj Ratanaworabhan, and Nana B Sam. The VPC trace-compression algorithms. *Computers, IEEE Transactions on*, 54(11):1329–1344, 2005.

[4] Martin Burtscher and Benjamin G Zorn. Exploring last n value prediction. In *Parallel Architectures and Compilation Techniques, 1999. Proceedings. 1999 International Conference on*, pages 66–76. IEEE, 1999.

[5] Alan Eustace and Amitabh Srivastava. Atom: A flexible interface for building high performance program analysis tools. In *Proceedings of the USENIX 1995 Technical Conference Proceedings*, pages 25–25. USENIX Association, 1995.

[6] Freddy Gabbay and Avi Mendelson. *Speculative execution based on value prediction*. Citeseer, 1996.

[7] Bart Goeman, Hans Vandierendonck, and Koenraad De Bosschere. Differential fcm: Increasing value prediction accuracy by improving table usage efficiency. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 207–216. IEEE, 2001.

[8] A.B.T. Hopkins and K.D. McDonald-Maier. Debug support strategy for systems-on-chips with multiple processor cores. *Computers, IEEE Transactions on*, 55(2):174–184, Feb 2006.

[9] Nios II and Cyclone III. Nios II Processor Reference Handbook Section 2, Processor Architecture. 2003.

[10] MIPS Technologies Inc. The PDtrace interface and trace control block specification. 2002.

[11] Infineon. TC1775 User's Manual, System Units, Section 20. 2001.

[12] S. Kanev and R. Cohn. Portable trace compression through instruction interpretation. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 107–116, April 2011.

[13] Chung-Fu Kao, Shyh-Ming Huang, and Ing-Jer Huang. A hardware approach to real-time program trace compression for embedded processors. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(3):530–543, March 2007.

[14] Ciaran MacNamee and Donal Heffernan. Emerging on-chip debugging techniques for real-time embedded systems. *Computing & Control Engineering Journal*, 11(6):295–303, 2000.

[15] Eric Rotenberg, Steve Bennett, and James E Smith. A trace cache microarchitecture and evaluation. *Computers, IEEE Transactions on*, 48(2):111–120, 1999.

[16] Bohuslav Rychlik, John Faistl, Bryon Krug, and John Paul Shen. Efficacy and performance impact of value prediction. In *Parallel Architectures and Compilation Techniques, 1998. Proceedings. 1998 International Conference on*, pages 148–154. IEEE, 1998.

[17] Yiannakis Sazeides and James E Smith. The predictability of data values. In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, pages 248–258. IEEE, 1997.

[18] Colleen D. Schieber and Eric E. Johnson. Ratchet: Real-time address trace compression hardware for extended traces. *SIGMETRICS Perform. Eval. Rev.*, 21(3-4):22–32, April 1994.

[19] Dean M Tullsen and John S Seng. Storageless value prediction using prior register values. In *ACM SIGARCH Computer Architecture News*, volume 27, pages 270–279. IEEE Computer Society, 1999.

[20] V. Uzelac, A. Milenkovic, M. Milenkovic, and M. Burtscher. Using branch predictors and variable encoding for on-the-fly program tracing. *Computers, IEEE Transactions on*, 63(4):1008–1020, April 2014.

[21] Robert NM Watson, Jonathan Woodruff, W Simon, Steven J Murdoch, Peter G Neumann, Robert Norton, and Michael Roe. Bluespec Extensible RISC Implementation (BERI): Hardware reference. *University of Cambridge, Computer Laboratory, Technical Report*, (UCAM-CL-TR-852), 2014.