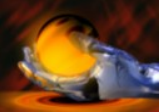


Maximizing multicore memory bandwidth using compression

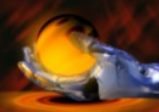


Lawrence Spracklen
Sun Microsystems



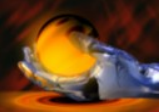
Overview

- Bandwidth challenges – motivation
- Compression opportunities
- Compression algorithms & results
- Conclusions



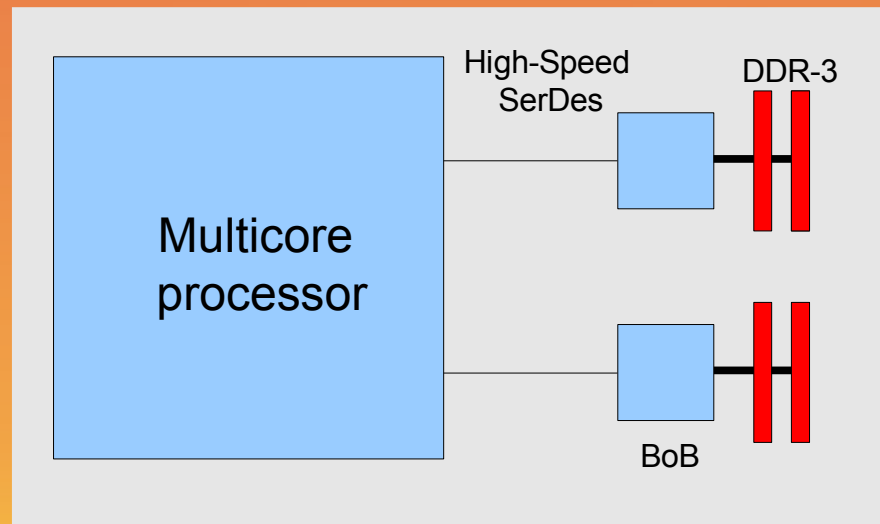
Acknowledgements

- Some of this research is published in IEEE Transactions on Computers
 - Joint paper with Martin Thuresson & Per Stenstrom (Chalmers University, Sweden)

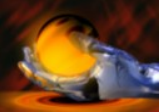


High-end multicore processors

- High performance multicore processors no longer directly connect to DDR memory



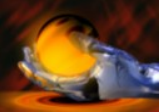
- High-speed SerDes links off-chip to BoBs



Motivation

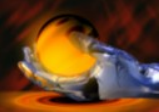
- Latest multicore design point has many cores per chip and modest on-chip caches
- Significant offchip bandwidth (BW) requirements
 - Per generation core doubling has rapidly increased BW consumption
- Pin limitations can constrain options for scaling BW
- Ramping of SerDes frequencies is helping
- But increasing SerDes frequency has power ramifications
 - Offchip links (mem+coherency) can now burn significant power
- Performance can become bounded by off-chip BW
- Can we do more with the available resources?

=> Leverage compression over SerDes



Additional benefits of compression

- **Hide decompression penalty:** can speculatively compress/decompress cache line in parallel with ECC computation, hiding/reducing overhead
 - Can retain data in compressed form offchip – no need for inbound compression
- **Reduce latency:** transmitting cache line over narrow serial links can represent a not insignificant proportion of the total latency of an offchip miss
 - By reducing amount of data transmitted, compression can noticeably reduce the latency of offchip misses
- While compression is normally associated with increased latencies, in many situations, can not only hide compression overhead but can reduce overall latency!!



Compression considerations

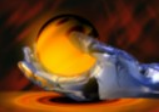
- Choice of whether to compress each cache line in isolation or retain inter-cache line compression state

Inter-cache line state

- Can yield significant improvements in compression
- Only feasible for point-to-point links
- Potentially complicated by link errors
- Not suited for use as cache/memory compression

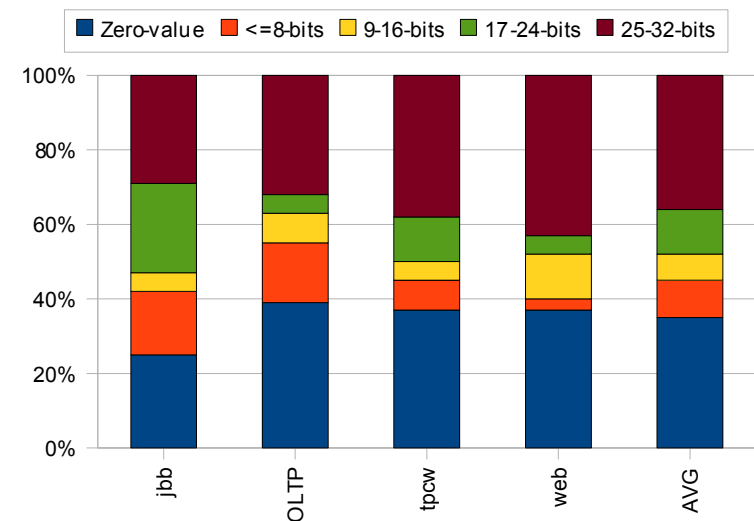
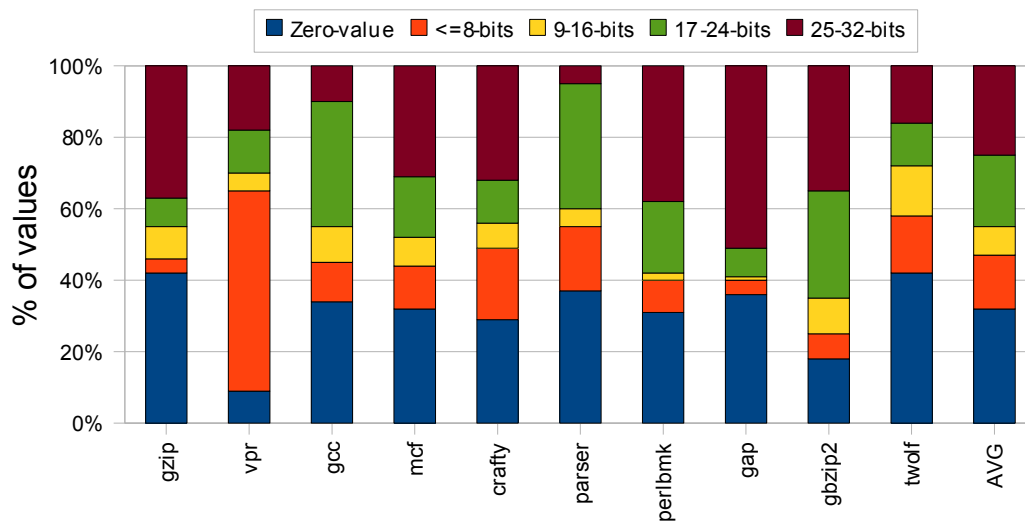
In-isolation 'stateless' compression

- Each cache line compressed in isolation
- For some schemes, can approximate benefits of inter-cache line state by prefilling history buffer

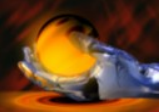


Value distribution

- Look at values transferred offchip
 - Most apps have a large number of very small integers and a large number of large integers (32-bit analysis)

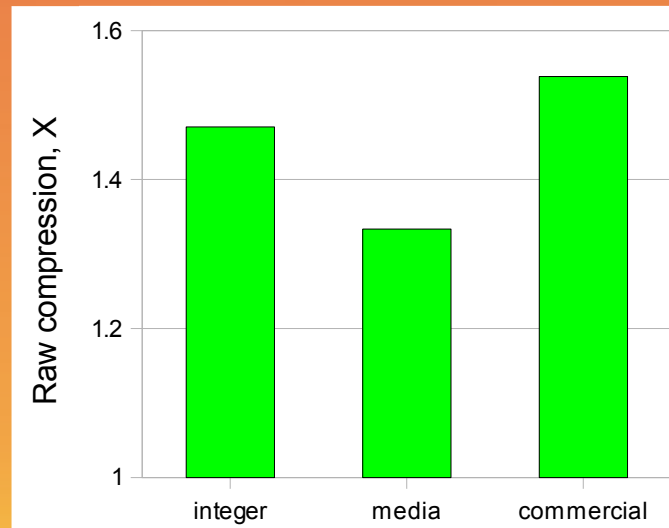


- Could just leverage frequent zero-values for compression

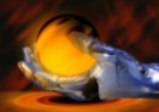


Dropping zeros

- Significant compression can be achieved
 - Dictionary schemes can deliver significantly higher compression, but too slow
 - Also need to worry about total channel compression

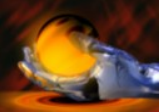


- Dropping zeros very simple, but can we achieve more while controlling complexity?



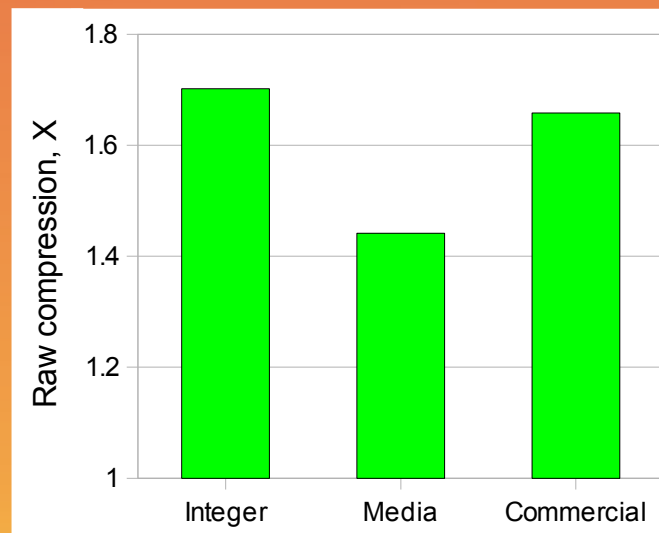
Small value locality

- Significance width compression efficiently encodes small integers
- Each encoded value consists of 2 parts:
 - The fixed-length prefix that indicates the length of the data field
 - The variable-length data field that contains the actual value
- For example, a 32-bit integer can be encoded using a 5-bit prefix and a variable length data field
 - e.g. 0X000000009 can be encoded as 0x0899
- Can reduce size of prefix by constraining permissible lengths of data field
 - e.g. Partition data values into 32, 24, 16, and 8-bit lengths – only needs a 2-bit prefix

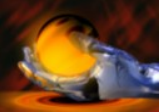


Small value locality performance

- As expected, exploiting SVL delivers significant savings
 - Limiting data lengths (& reducing prefix size) generally provides best result

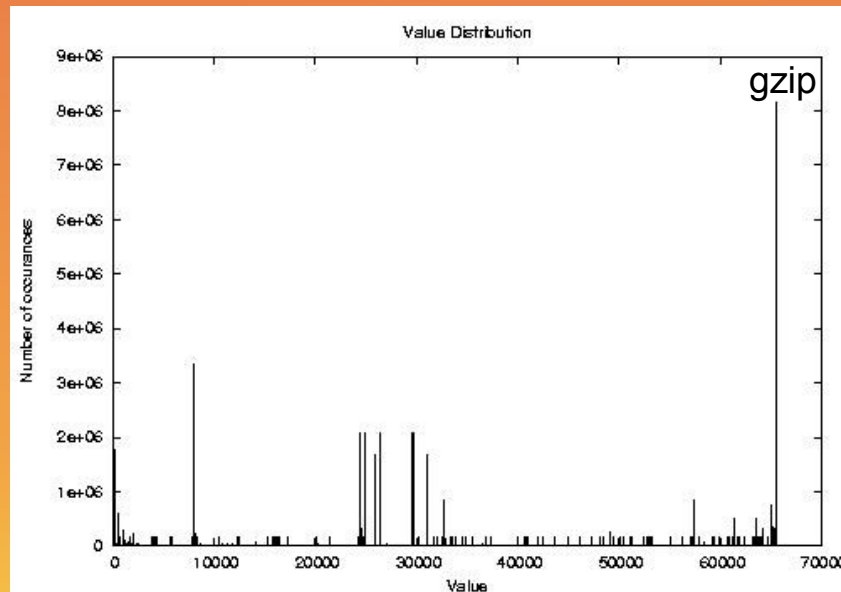


- Decent improvement over the basic drop zeros scheme
 - Also more robust

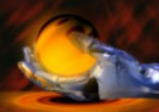


Clustered value locality

- In addition to frequent small integers, applications often exhibit additional clustering in remaining value space
 - i.e. very uneven value distribution

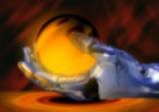


- Efficiently encode values via a reference to a representative value for each cluster



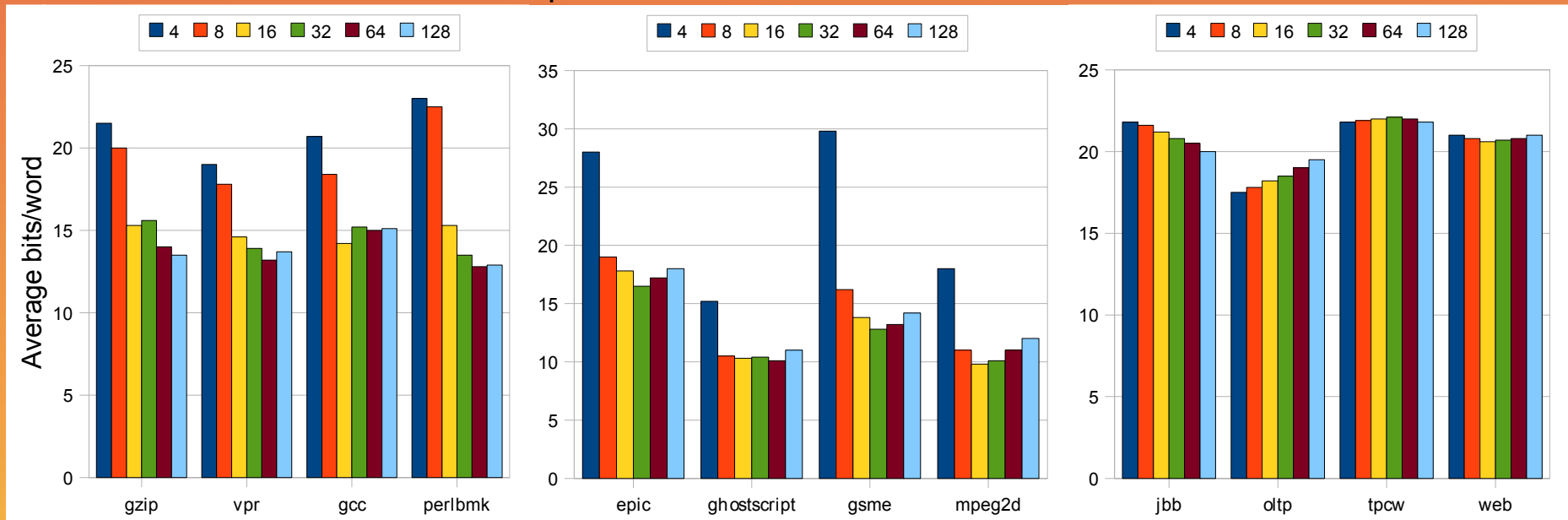
Delta encoding

- Given clustering, can efficiently encode values as
 - **Fixed-length index** : indicates which representative value to use
 - **Variable length delta field** : difference between reference value and encoded value
- Implemented using a cache of representative values at either side of link
 - Cache is scanned for closest match
 - Index of closed match & delta is transmitted if delta is small (& cache updated)
 - If difference is larger than threshold send value and replace LRU cache element
- Efficiently adapts as old clusters are gradually replaced with new clusters
- Trade-off between benefits of larger cache and drawbacks of larger index

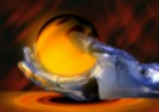


Delta encoding performance

- Investigated a wide range of value cache sizes
 - Index overheads become prevalent as cache size increases

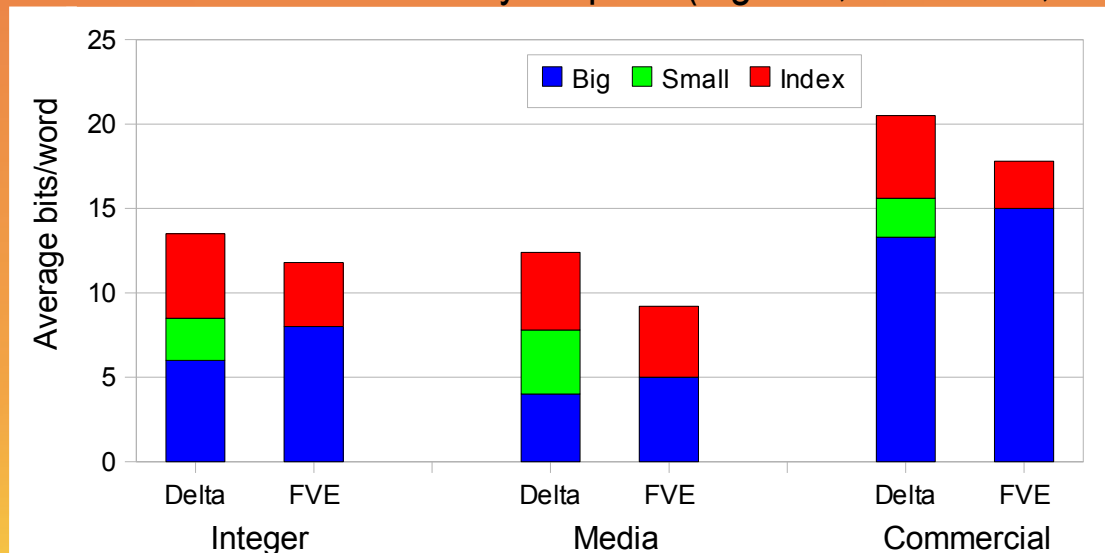


- Performance generally not as good SWC results
- Experimented with various replacement thresholds etc.

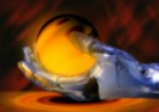


Frequent value encoding

- In contrast to sending delta from representative value, only send exact matches
 - More precise encoding of exact matches than achieved for delta encoding
 - Great if certain values are very frequent (e.g. 0x0, 0xbadcafe, 0xdeadbeef etc)

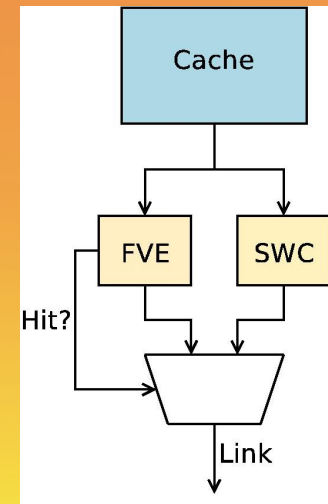
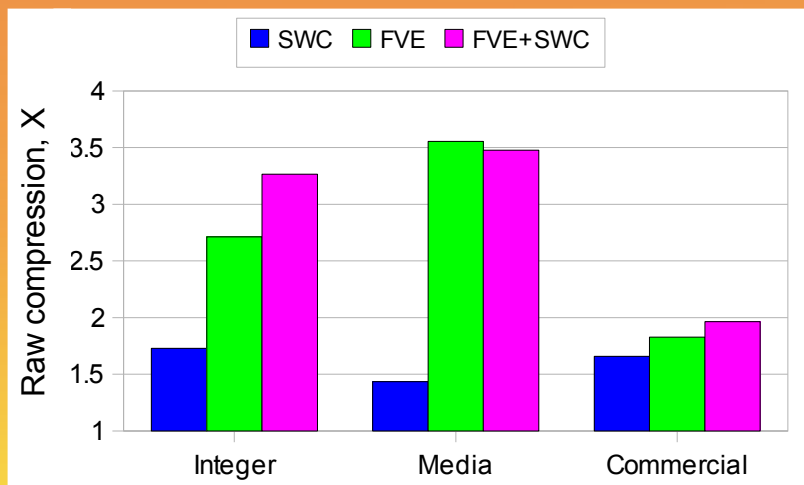


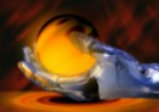
- Outperforms delta encoding for all 3 application categories



Combined approaches

- Weakness of FVE lies in cost of transferring misses
- SWC will help if a number of the misses are small values
- Combine both techniques
 - Hit in FV\$ - transfer index (don't need to bother with small integers – use SWC)
 - Miss in FV\$ - replace LRU FV\$ entry and send miss SWC encoded





Conclusions

- Offchip bandwidth (BW) requirements of multicore processors are significant
- Compression can be utilized to provide significant improvement in the effective offchip BW
 - Or allows significant reductions in SerDes lanes or frequency (power/cost savings)
- Even simple compression schemes yield significant benefits for key applications
 - Increases effective offchip BW by up to 3X
- Compression can potentially be leveraged without the expected latency impact



Questions?