Application-Based Fault Tolerance Techniques for Fully Protecting Sparse Matrix Solvers

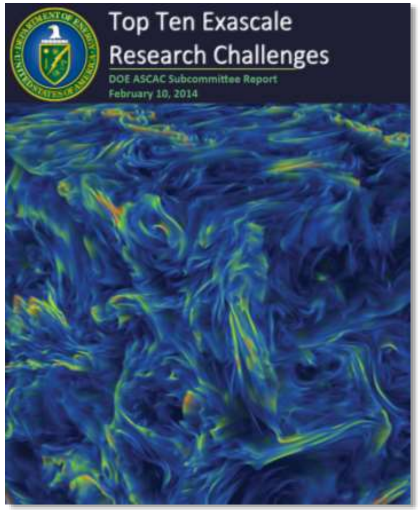
Grzegorz Pawelczak, Simon McIntosh-Smith, James Price, Matt Martineau

University of Bristol - High Performance Computing Group

https://uob-hpc.github.io

Top 10 Exascale challenges

1. **Energy efficiency**: Creating more energy-efficient circuit, power, and cooling technologies.
2. **Interconnect technology**: Increasing the performance and energy efficiency of data movement.
3. **Memory technology**: Integrating advanced memory technologies to improve both capacity and bandwidth.
4. **Scalable system software**: Developing scalable system software that is power- and resilience-aware.
5. **Programming systems**: Inventing new programming environments that express massive parallelism, data locality, and resilience
6. **Data management**: Creating data management software that can handle the volume, velocity and diversity of data that is anticipated.
7. **Exascale algorithms**: Reformulating science problems and redesigning, or reinventing, their solution algorithms for exascale systems.
8. **Algorithms for discovery, design, and decision**: Facilitating mathematical optimization and uncertainty quantification for exascale discovery, design, and decision making.
9. **Resilience and correctness** Ensuring correct scientific computation in face of faults, reproducibility, and algorithm verification challenges.
10. **Scientific productivity**: Increasing the productivity of computational scientists with new software engineering tools and environment



February 2014

University of BRISTOL

Why do we need FT?

- Many different kinds of fault can occur during computation (G. Gibson, Proc. of the DSN2006, June, 2006):
  - Soft errors (bit flips in memory etc)
  - Hard errors (component breakage)
  - Power outages
  - OS errors
  - System software errors
- In this work we're interested in the faults which affect the program data

University of BRISTOL

Current Solutions

- Error Correcting Codes implemented in hardware
- Common Codes:
  - Parity (SED)
  - Hamming code - Single Error Correction and Double Error Detection (SECDED)
  - Reed–Solomon code - Chipkill
- ECC does not come for free!
  - Storage overhead
  - Extra energy and bandwidth used
  - Puts restrictions on the hardware that can be used

University of BRISTOL

Error Detecting/Correcting Codes

- Data is stored as **Codewords** in memory
- The codes we are interested in are
  - Parity - 1 extra bit per codeword
  - Hamming Code - Single Error Correction and Double Error Detection (SECDED) with:
    - 64-bit codewords with 8-bits of redundancy
    - 128-bit codewords with 9-bits of redundancy
  - Cyclic Redundancy Check (CRC) Code
    - In particular CRC32C with 32-bits of redundancy per codeword

University of BRISTOL

Application Based Fault Tolerance

- Can take advantage of the data structures and memory access patterns of the application
- User knowledge enables wider range of fault recovery techniques
- A lot of progress being made in:
  - Dense linear algebra
  - Monte Carlo
  - Sparse linear algebra (this work)
  - Spectral (FFT)

ABFT for Sparse Matrix Solvers

- In our research we utilise the TeaLeaf mini-app
  - Part of Sandia National Laboratories' Mantevo (https://mantevo.org/) mini-app benchmark suite
- TeaLeaf solves the linear heat conduction equation in 2D on a spatially decomposed regular grid using a five-point stencil
- Vast majority of TeaLeaf's runtime (+98%) is spent performing either matrix-vector products or dot products
- Two main data structures
  - Sparse matrix
  - Dense vectors

University of BRISTOL

Sparse Matrix Storage

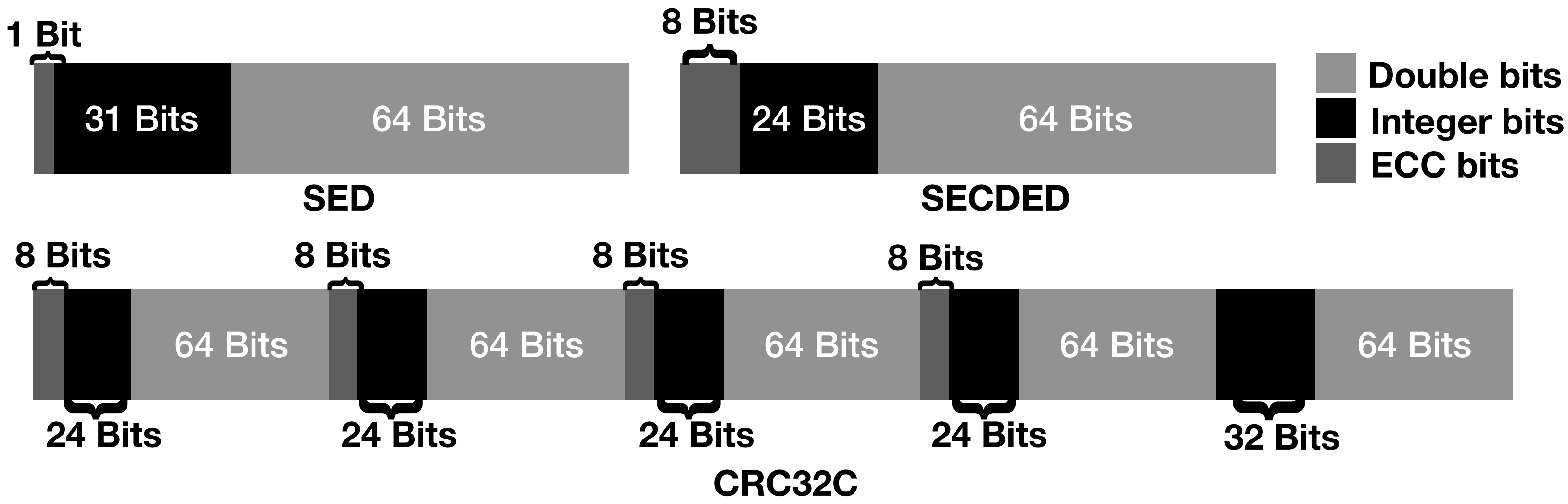
- Most of the matrix elements are zero
- To save space, usually stored in compressed formats
- We focus our efforts on the Compressed Sparse Row (CSR) format where a m×n matrix is represented by three dense vectors:
  - Vector **v** stores the corresponding nonzero values
  - Vector **c** stores the column indices for each non-zero value
  - Vector **r** stores the offsets of the first nonzero element in each row

ABFT with no storage overhead

- **Observation 1**: If the matrix has less than 2^{32} - 1 **columns**, then elements in the column vector **c** will have unused bits
- **Observation 2**: If the matrix has less than 2^{32} - 1 **nonzero values**, then elements in the row offsets vector **v** will have unused bits
- By further restricting the matrix size we can repurpose these unused bits to store the redundant ECC data
- Note that in many production solvers, the matrix dimensions may not meet our requirements, however:
  - These restrictions apply to a single process
  - Our techniques are easily extended to 64-bit integers

University of BRISTOL

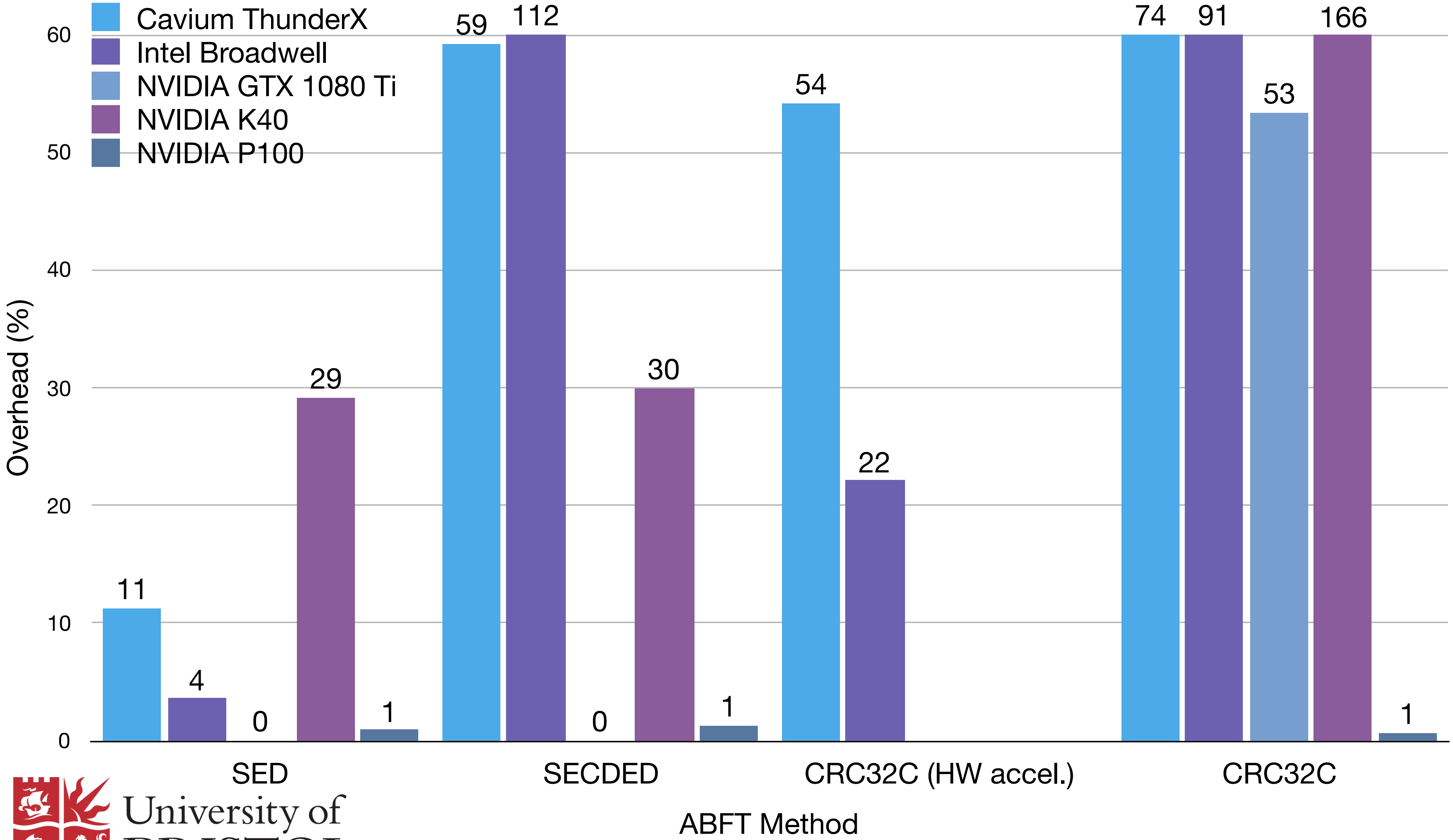
Protecting the CSR Elements



- A CSR element is formed by pairing a nonzero value from vector **v** with the corresponding column index from vector **c** to form a 96-bit CSR element
- This poses the following limits on the number of columns:
  - SED - maximum 2^{31} - 1 columns
  - SECDED or CRC32C - maximum 2^{24} - 1 columns
- When using CRC with a 32-bit checksum, we protect the whole matrix row at a time

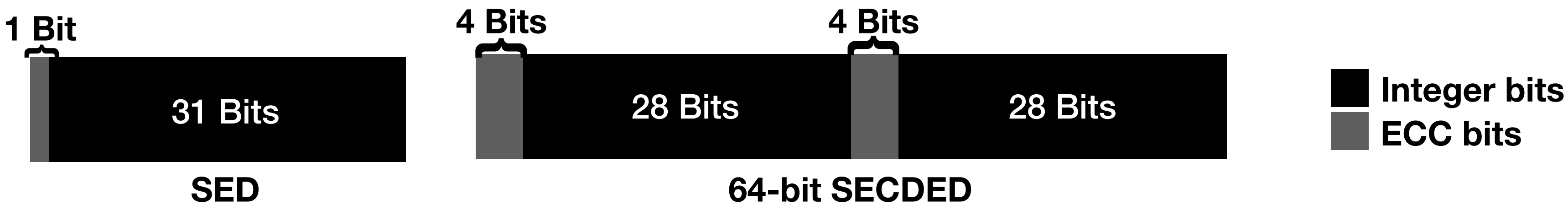
University of BRISTOL

Performance Results



University of BRISTOL

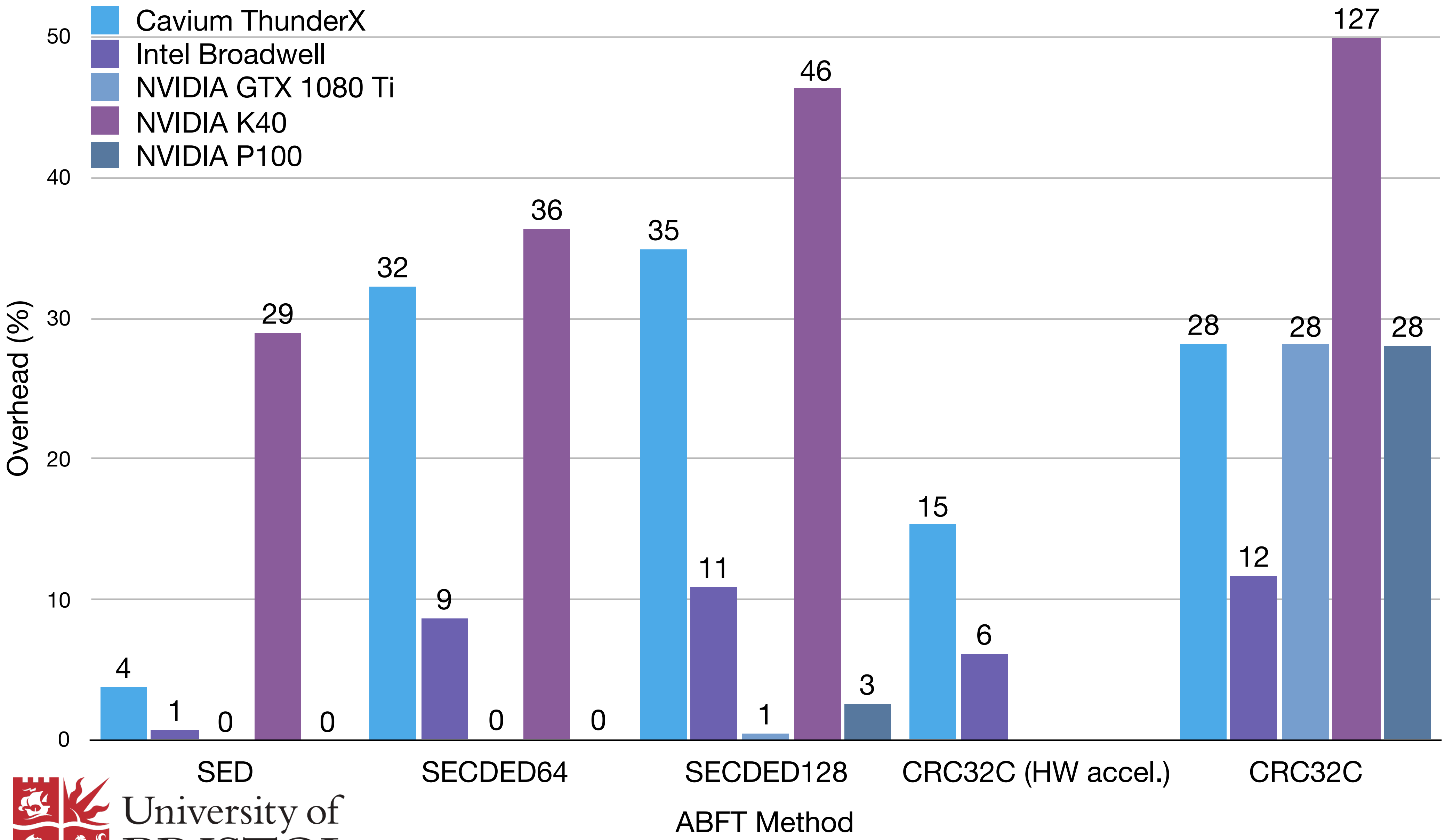
Protecting the Row Offset Vector



- A similar approach for protecting the CSR elements can be applied to protecting the **r** row offset vector
- When using SED:
  - Matrix can have at most 2^{31} - 1 nonzero elements.
- In order to use other ECC techniques, we use the top 4 bits from each elements
  - The matrix can still have 2^{28} - 1 or ≈ 268 million nonzero elements
- Other ECC techniques require more than 4 bits to store the redundancy
  - Protect multiple elements at the same time and split the redundancy bits between multiple elements

University of BRISTOL

Performance Overheads



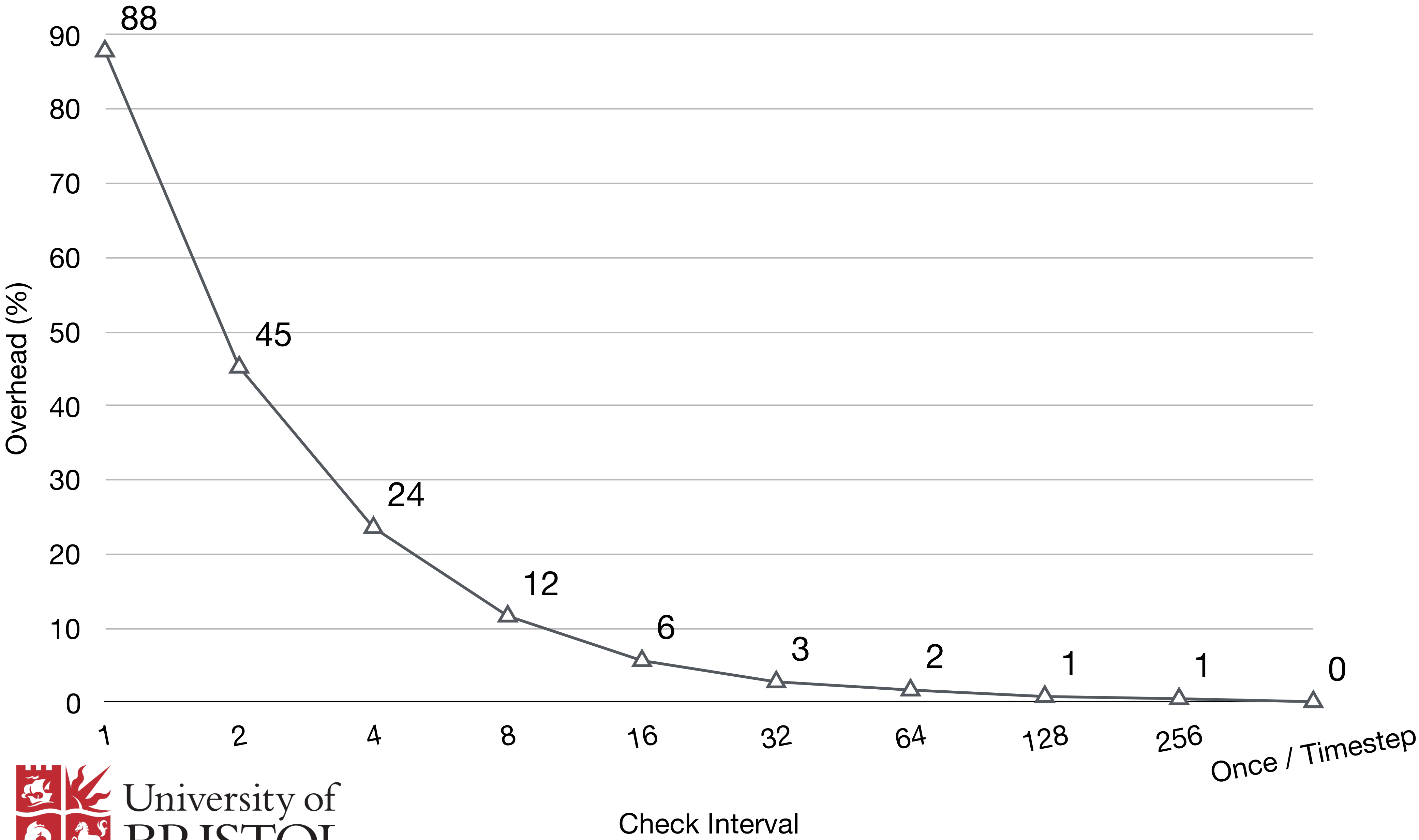
University of BRISTOL

Less Frequent Checking

- Observation: During the Conjugate Gradient Solve the matrix does not change
- Perform the matrix integrity checks every N iterations of the algorithm
- Boundary checks on the column and row vector are performed to prevent out of bounds memory access
- Now have to perform up to N more iterations of CG before the error is detected
- We are not able to fully correct any errors, only detect

University of BRISTOL

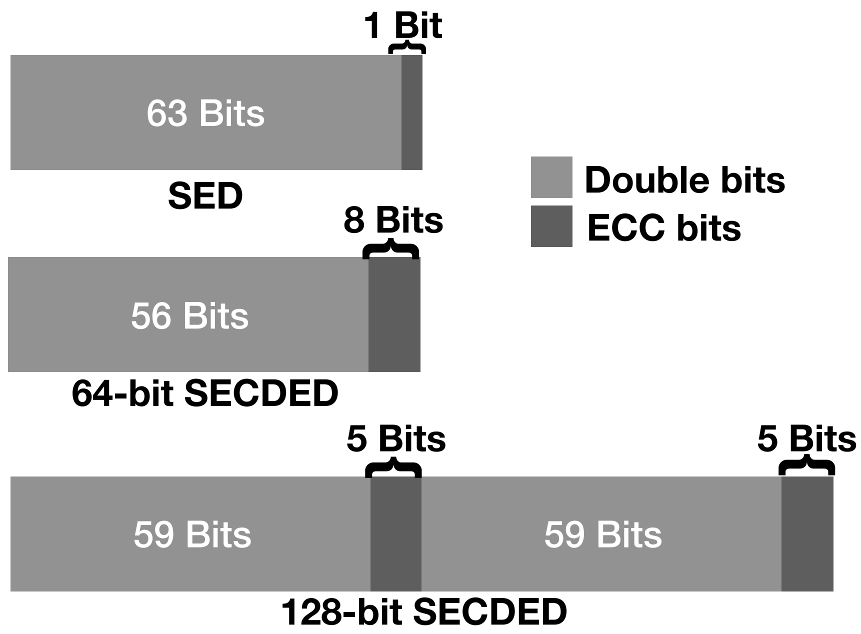
Performance Results



University of BRISTOL

Floating Point Vector Protection

- Floating point values do not have any unused bits due to their format
- Redundancy bits are stored in the least significant bits of the mantissa
- This storage method poses a risk that the solver may take longer to converge or fail to converge altogether
- The solver has always converged, with the norm of the solution vector within $2.0 \times 10^{-11}\%$ of the expected answer
- Increase in the total number of iterations was less than 1%



University of BRISTOL

Read-Modify-Writes

- Unlike the sparse matrix, the floating point vectors change their values
- When modifying a value in a vector, a Read-Modify-Write (RMW) has to be performed as only part of the codeword is being modified
    - Results in two ECC calculations every write
- Concurrency issues when multiple processes try to write the same ECC codeword

University of BRISTOL

Avoiding RMWs

- **Observation:** When performing calculations at position i, the algorithm will then work on the next element at position i + 1
- By buffering the writes a whole ECC element can be committed to memory in one go
  - Single ECC calculation per multiple writes
- The algorithm has to be adapted so that the calculations are performed on the whole ECC element at a time
- Removes the race conditions

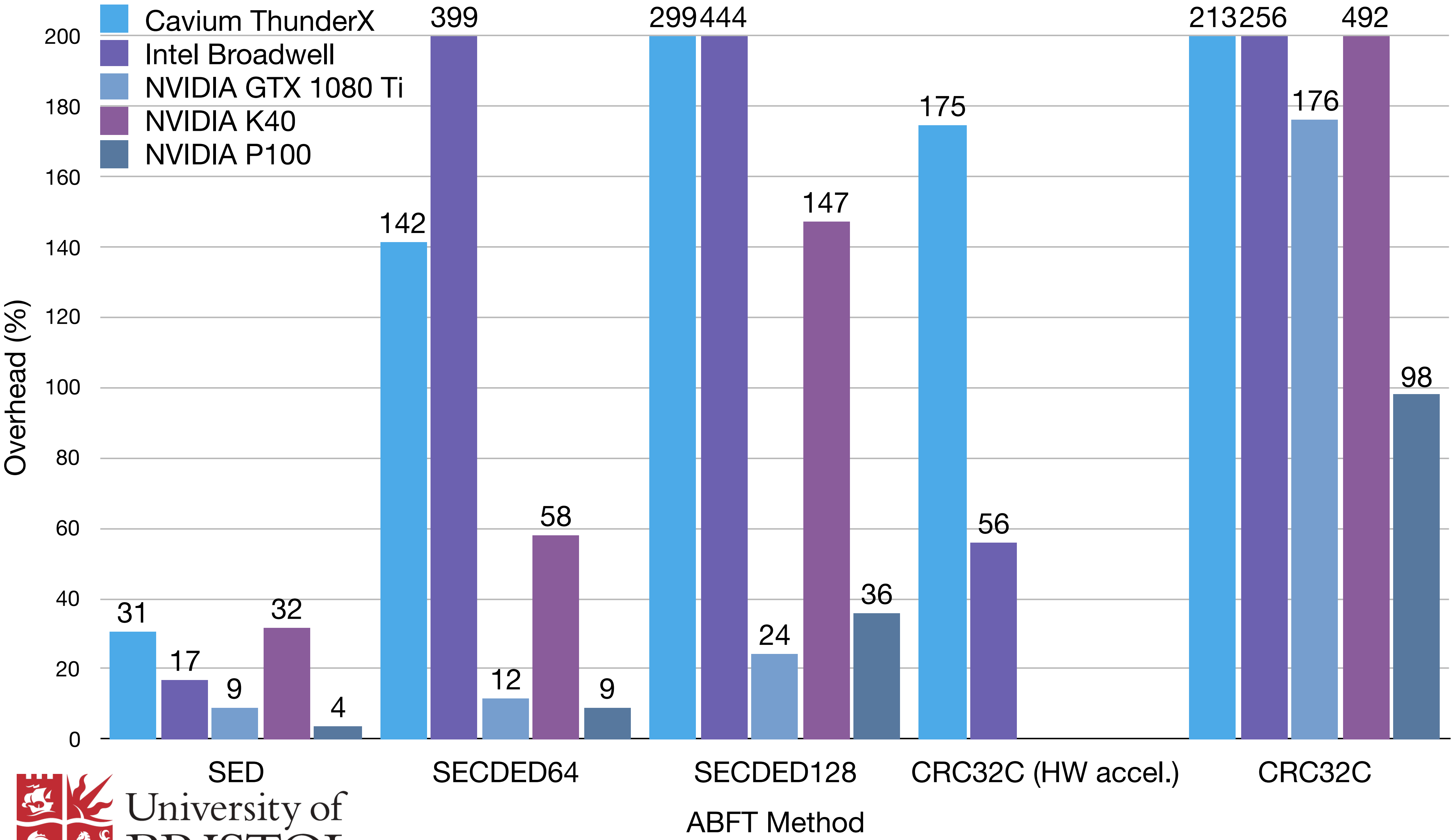
University of BRISTOL

Caching

- By buffering ECC elements when performing reads most of duplicate integrity checks can be removed
- This buffering technique performs poorly for the Sparse Matrix - Vector multiplication due to five-point stencil access pattern
  - At least 3 ECC compound elements are accessed per iteration
- By leveraging the knowledge about the application we can create a caching scheme within the kernel that is both multiple ECC element and multi-iteration aware

University of BRISTOL

Performance Results



University of BRISTOL

Conclusions

- Demonstrated efficient ABFT techniques with no storage overhead
- We have shown that hardware accelerated calculations were a big improvement over software-only solutions
  - Instruction set design can help with achieving better performance, and that combining software and hardware methods to protect against errors might prove beneficial.
- Ideally these ABFT techniques would be implemented directly inside of libraries/packages such as **PETSc** or **Trilinos**

University of BRISTOL

References

- R. Hunt and S. McIntosh-Smith, **"Exploiting Spatial Information in Datasets To Enable Fault Tolerant Sparse Matrix Solvers"**, FTS, IEEE Cluster, Chicago, Sep 8^{th} 2015
- S. McIntosh-Smith, R. Hunt, J. Price and A. Vesztrocy, **"Application-Based Fault Tolerance Techniques for Sparse Matrix Solvers"**, to appear in IJHPCA, 2016
- J. Yeh, G. Pawelczak, J. Sewart, J. Price, A. Avila Ibarra, S. McIntosh-Smith, L. Bautista-Gomez, and F. Zyulkyarov, **"Software-level Fault Tolerant Framework for Task-based Applications"**, *Poster session, IEEE/ACM SuperComputing, Salt Lake City, United States*, 2016.

University of BRISTOL

Thank you!

Any questions?

University of BRISTOL

UOB HPC