Computational Significance
(and its implications for HPC)

Dimitrios S. Nikolopoulos
CCDSC
Dareizé, Oct. 5 2016

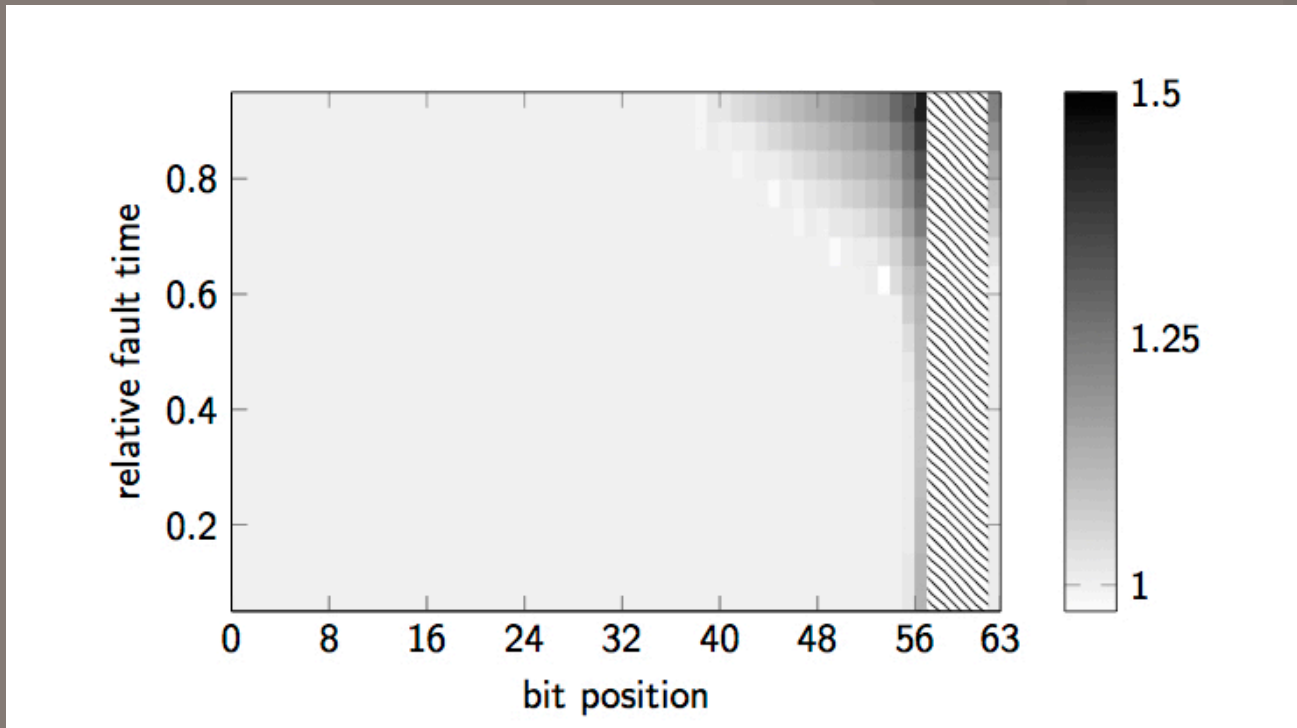
Challenge

- Transistors
  - ✓ Aggressive shrinking
  - ✓ Variation in performance, data retention times
- Two approaches
  - ✓ Mitigate it, lose performance
  - ✓ Embrace it, gain performance, introduce errors
- Best effort computing
  - ✓ Where algorithms are inherently approximate
  - ✓ Where algorithms or systems can mitigate errors

Significance-Driven Computing

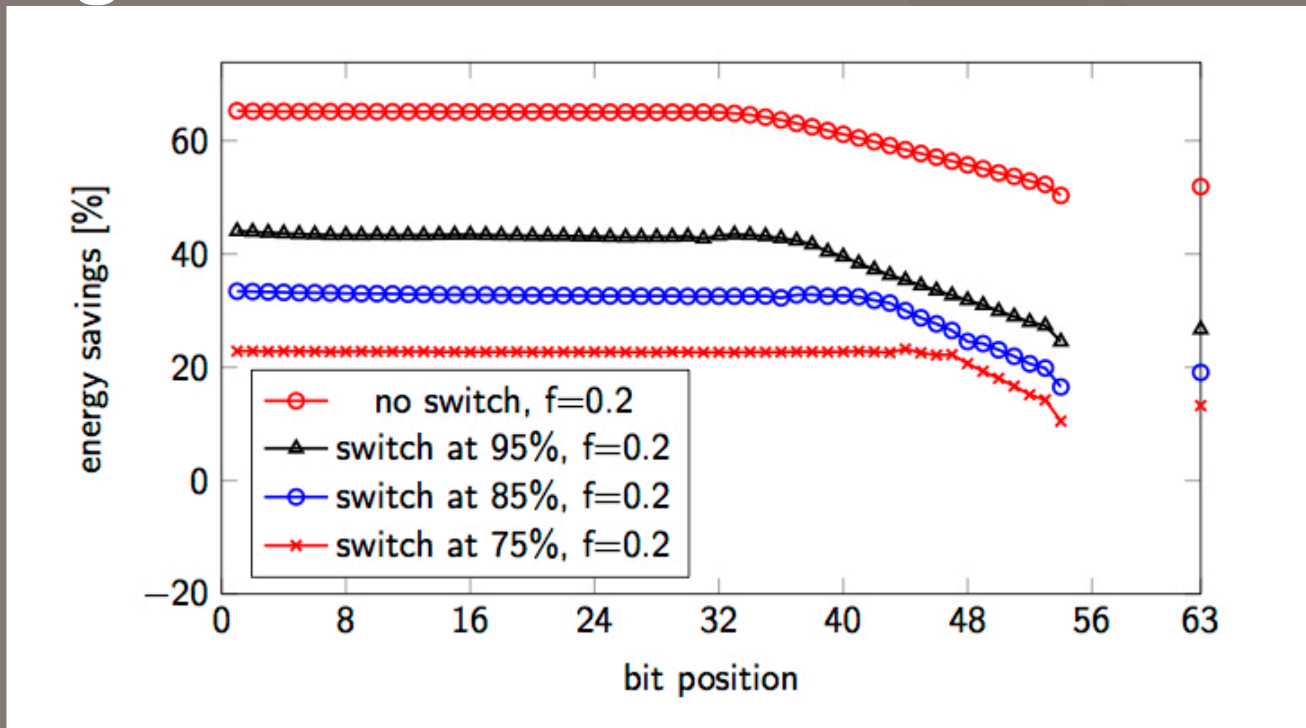
- Not every line of code or variable are equal
  - ✓ Each has a unique contribution to the output
  - ✓ Estimating this contribution needs domain expertise
- Computational significance
  - ✓ Value of contribution to output
- Disciplined approximation
- Abstraction for software
  - ✓ Selectively protect execution
    - ❑ memory objects, tasks, threads
  - ✓ Control error in the compiler, runtime, language
  - ✓ Algorithm complexity control

GMRES Resilience



Gscwandtner et al., CSR&D, 2015

Significance-driven GMRES

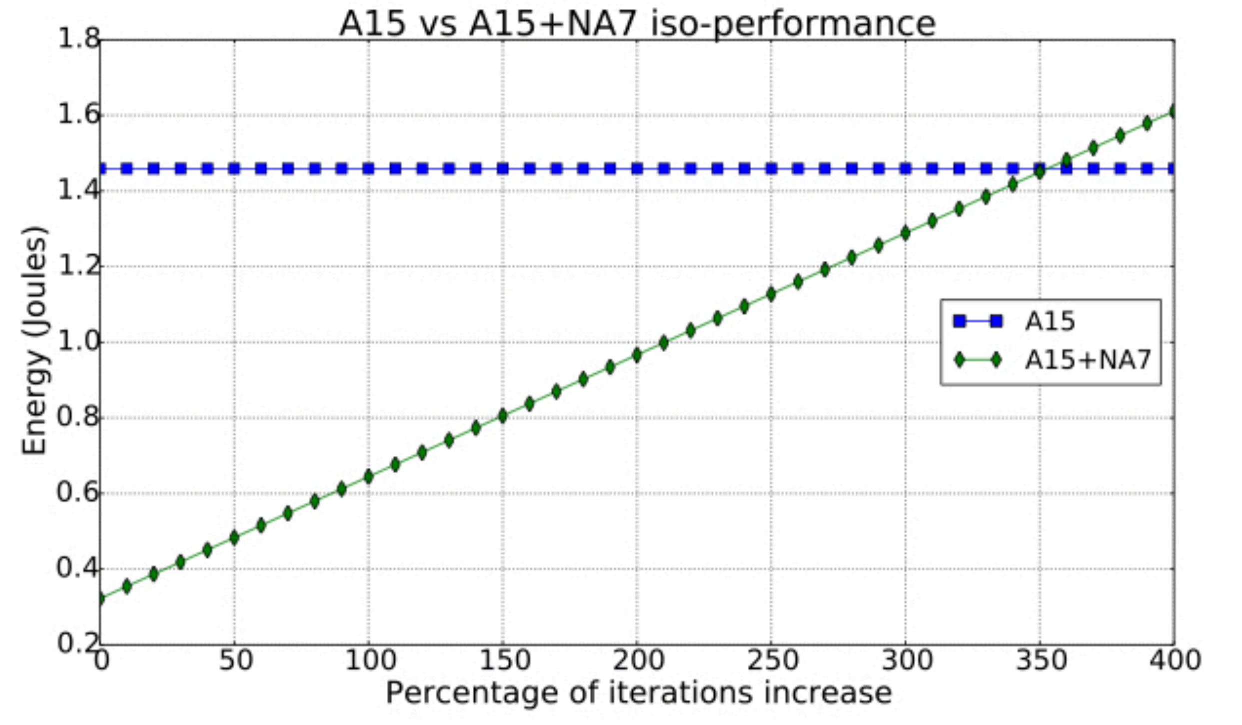


Vassiliadis et al., IJPP, 2016
Chalios et al., CDT, 2015

WE ARE QUEEN'S UNIVERSITY BELFAST We are exceptional

Queen's University Belfast

Self-stabilizing CG



- Algorithmic fault correction
  - ✓ Periodic step correcting state of algorithm
  - ✓ Guaranteed convergence with accurate healing step
  - ✓ No assumptions about convergence rate
- Heterogeneous architecture
  - ✓ 1-N reliable-unreliable core
  - ✓ Designed with iso-efficiency metrics
  - ✓ Healing step on reliable core

Aliaga et al., PARCO, 2015

Language & runtime support

- Disciplined approximation
  - ✓ User controls significance, error, performance
- Significance abstraction of code & data
  - ✓ Binary
  - ✓ Continuous
- Approximate alternatives of code blocks
- Examples
  - ✓ OpenMP tasks
    - ❑ Significance ‘score’, task alternatives
  - ✓ Dataflow annotations
    - ❑ Data criticality
  - ✓ App-specific error checks

Programming Model

```
double sobel(void) {
  int i;
  byte img[WIDTH*HEIGHT], res[WIDTH*HEIGHT];
  /* Initialize img array and reset res array */
  ...
  for (i=1; i<HEIGHT-1; i++)
    #pragma omp task label(sobel) approxfun(row_appr) \
      in(img[i*WIDTH:(i+1)*WIDTH-1]) \
      out(res[i*WIDTH:(i+1)*WIDTH-1]) \
      significant((i%9 + 1)/10.0)
    row_acc(res, img, i); /* Compute a single
                             output image row */
  #pragma omp taskwait label(sobel) ratio(0.35)
}
```

Aliaga et al., PARCO, 2015

Simple example: Convolution

```
/* sblY and sblY_appr are similar */
void row_acc(byte *res, byte *img, int i) {
  unsigned int p, j;
  for (j=1; j<WIDTH-1; j++) {
    p = sqrt(pow(sblX(img, i, j),2) +
             pow(sblY(img, i, j),2));
    res[i*WIDTH + j] = (p > 255) ? 255 : p;
  }
}

void row_appr(byte *res, byte *img, int i) {
  unsigned int p, j;
  for (j=1; j<WIDTH-1; j++) {
    /* abs instead of pow/sqrt,
       approximate versions of sblX, sblY */
    p = abs(sblX_appr(img, i, j) +
            sblY_appr(img, i, j));
    res[i*WIDTH + j] = (p > 255) ? 255 : p;
  }
}
```

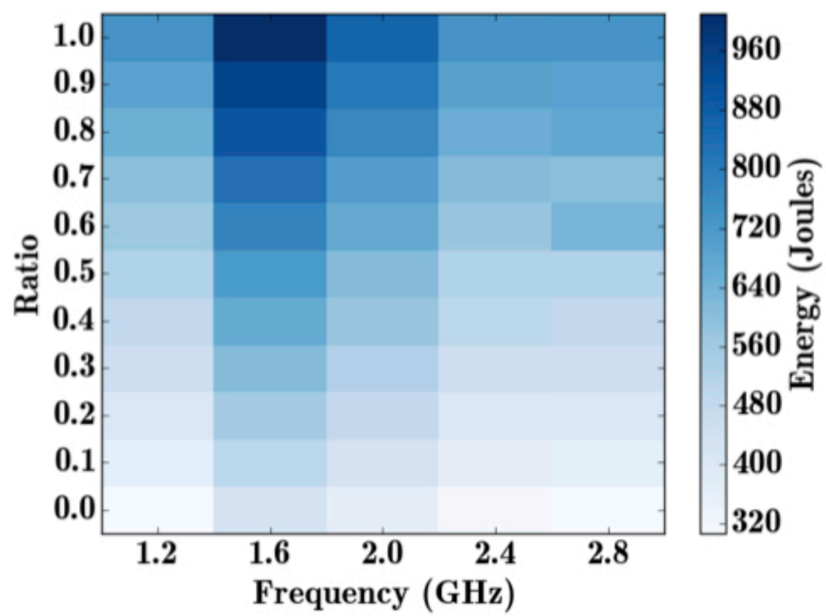
Aliaga et al., PARCO, 2015

Significance-driven runtime

- On-the-fly task versioning
  - ✓ Controlled approximation & error checking
- Quality-aware synchronization
  - ✓ Flimsy barriers
- Significance propagation
  - ✓ Track & tune significance of task groups & chains
- Multi-dimensional Optimization
  - ✓ Performance, Power, Energy, Quality

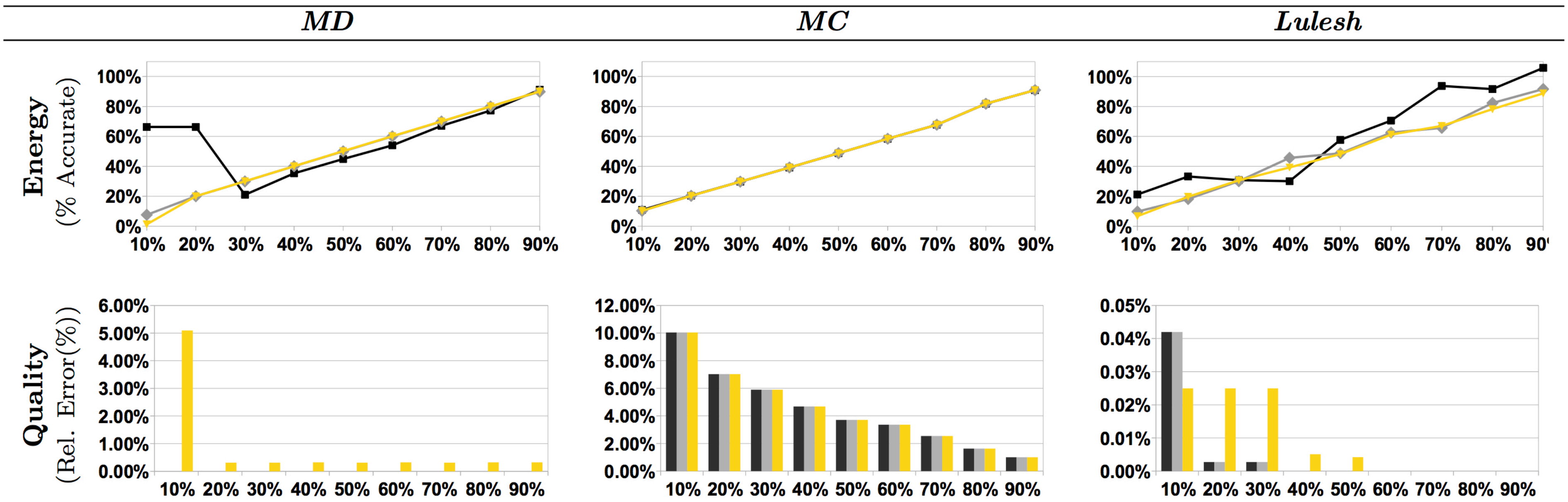
Vassiliadis et al., IJPP, 2016

Convolution trade-off's



Vassiliadis et al., CF, 2015
Vassiliadis et al., IJPP, 2016

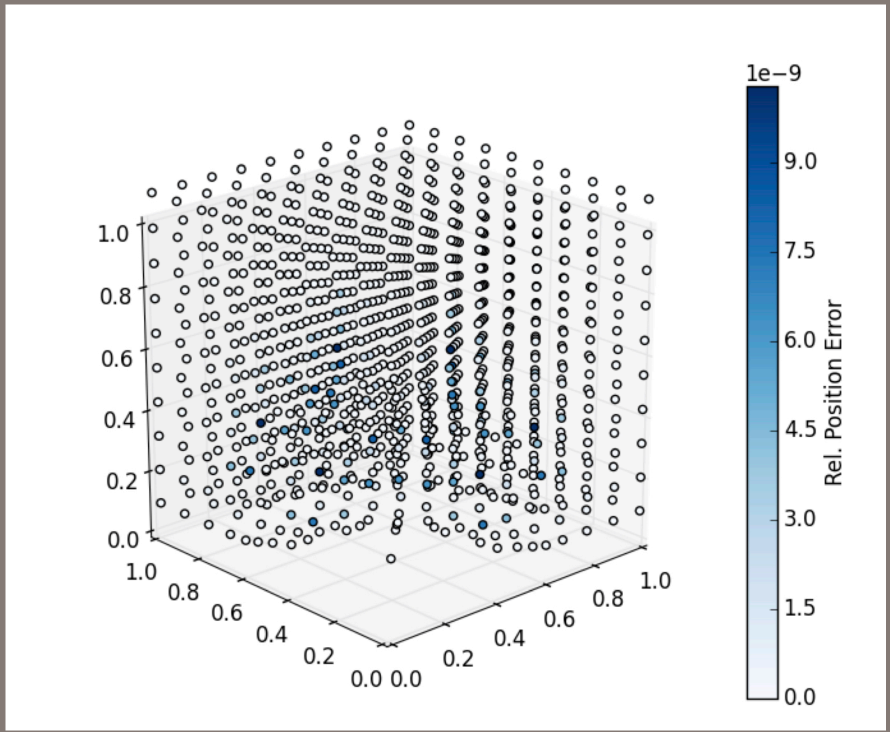
Some HPC app results



Vassiliadis et al., CF, 2015

Vassiliadis et al., IJPP, 2016

Lulesh error

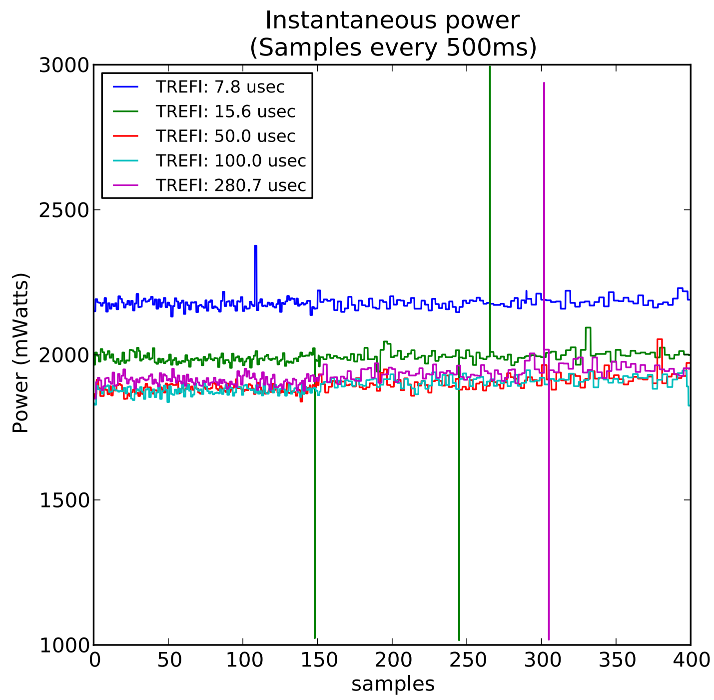


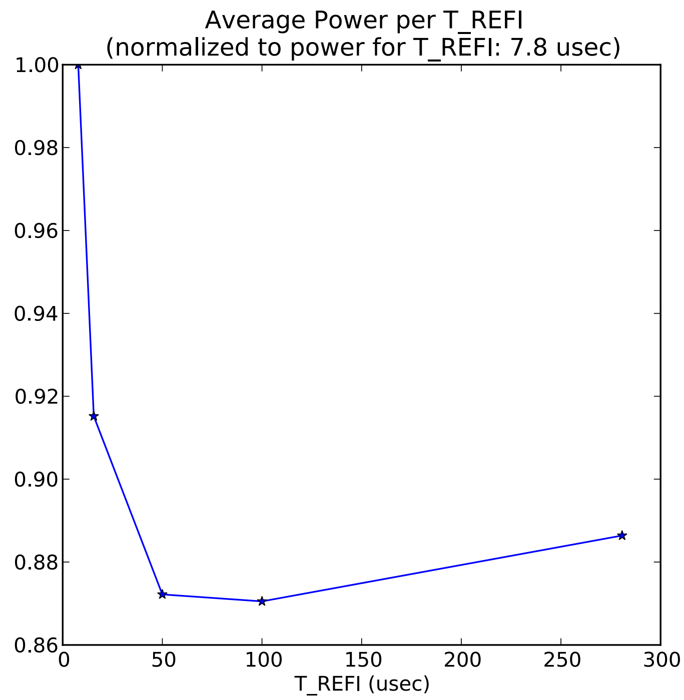
Vassiliadis et al., CF, 2015
Vassiliadis et al., IJPP, 2016

Variable-reliability memory

- DRAM refresh consumes significant power
  - ✓ Projected to 40%-50% in future large-memory systems
- Refresh-free memories
  - ✓ Additional errors
  - ✓ Many mitigation options (ECC, application)
- Significance-driven memory management
  - ✓ Data placement & migration
  - ✓ Memory reliability control

Variable-reliability memory

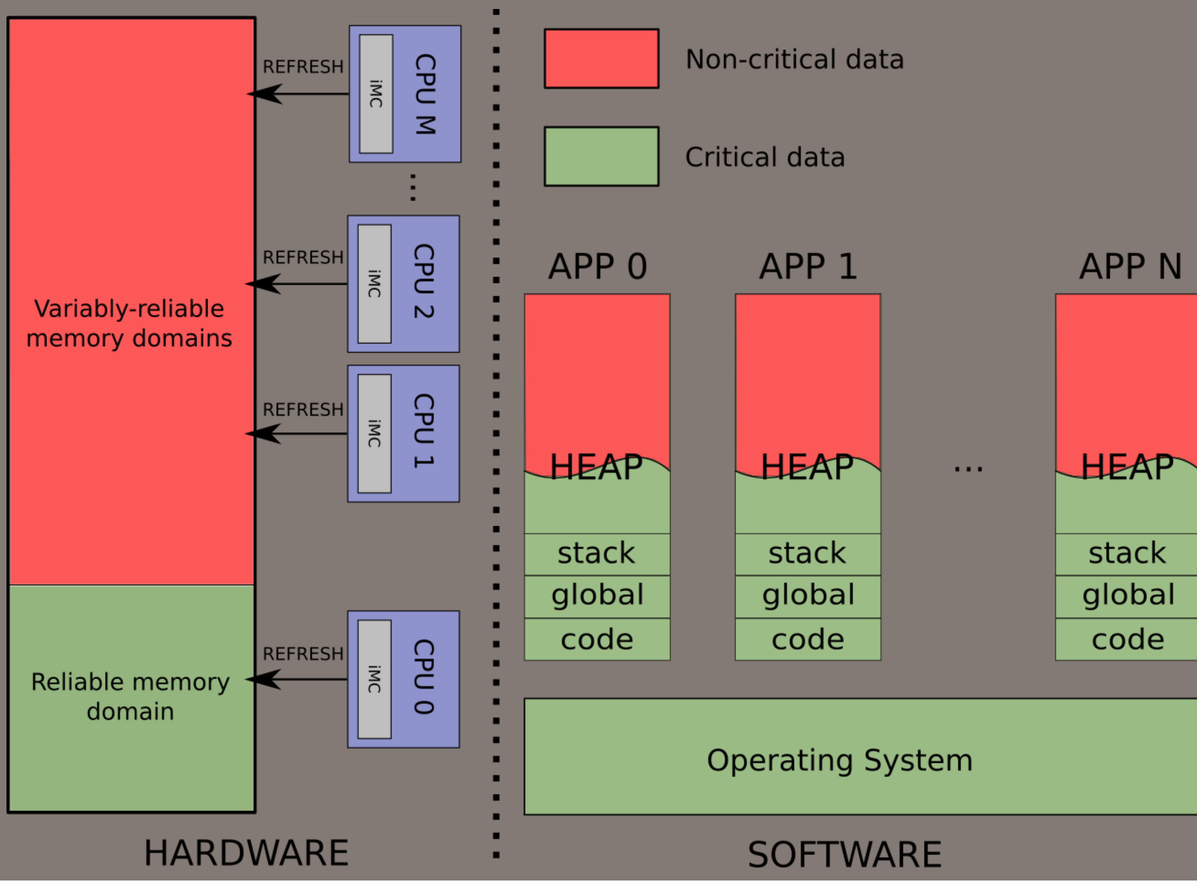




We are exceptional

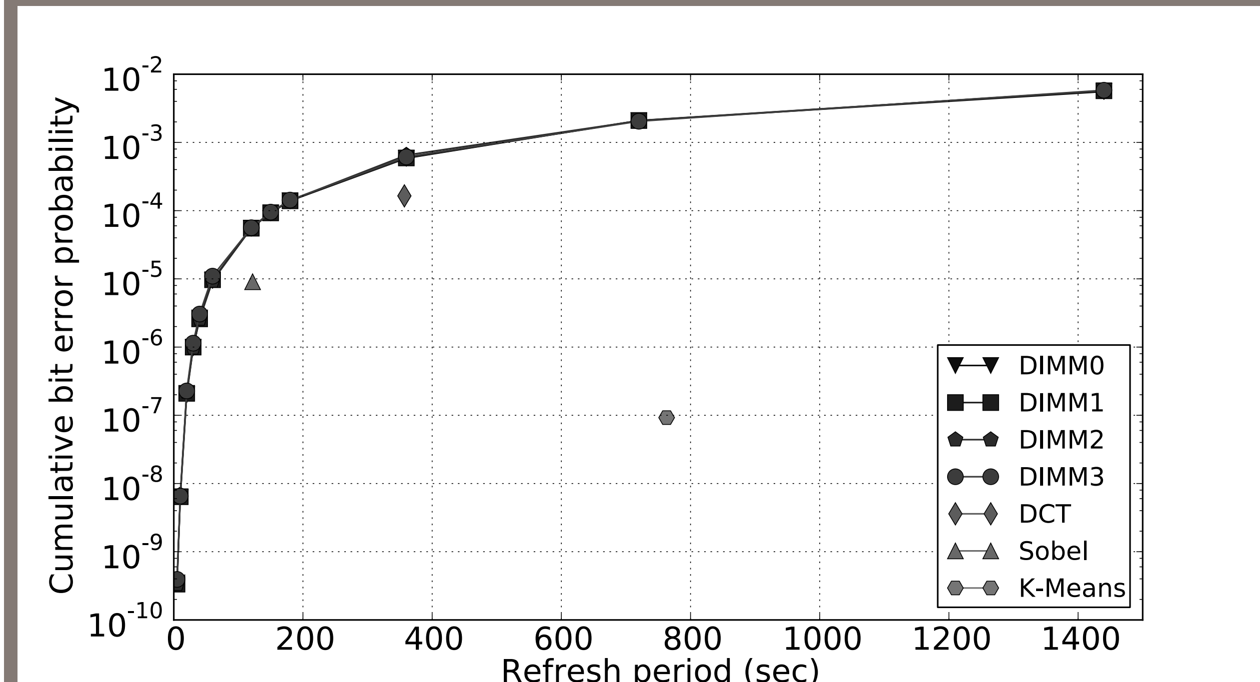
Queen's University Belfast

Relaxing refresh on an HPC server



- ✓ Divide physical memory to Reliable and Variably-Reliable Domains
- ✓ Allocate kernel to *RD*
- ✓ Allocate critical App data to *RD*
- ✓ Allow programmer to allocate heap to *VRD*

Application resilience



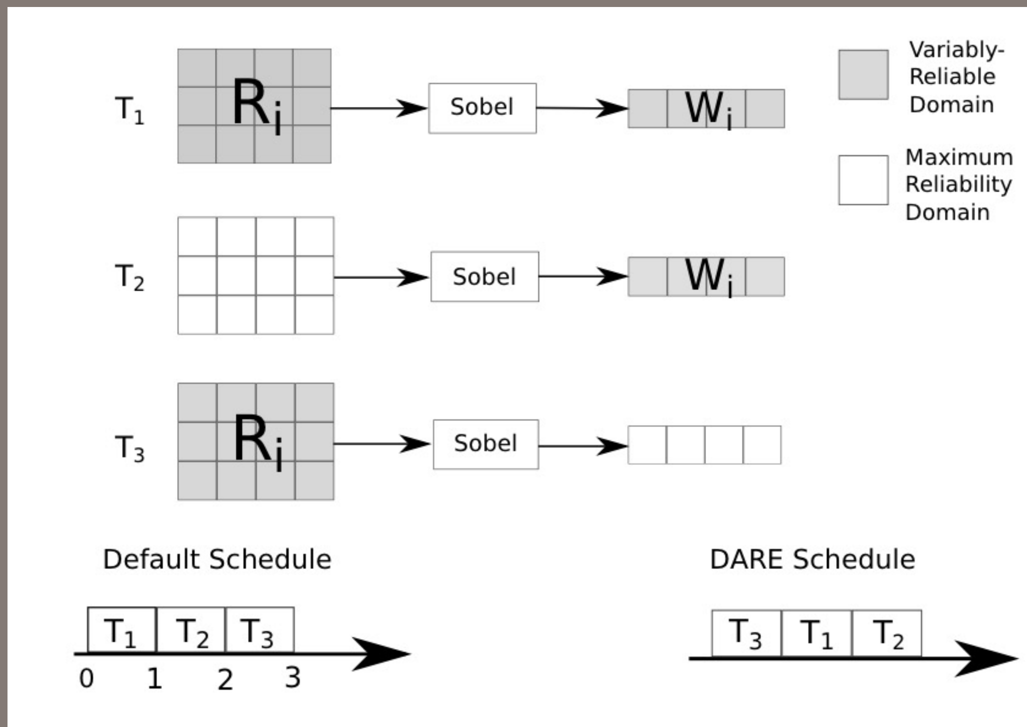
- ✓ Applications are naturally resilient, just by accessing data
- ✓ Potential for significant performance & energy gains

Application-level resilience methods

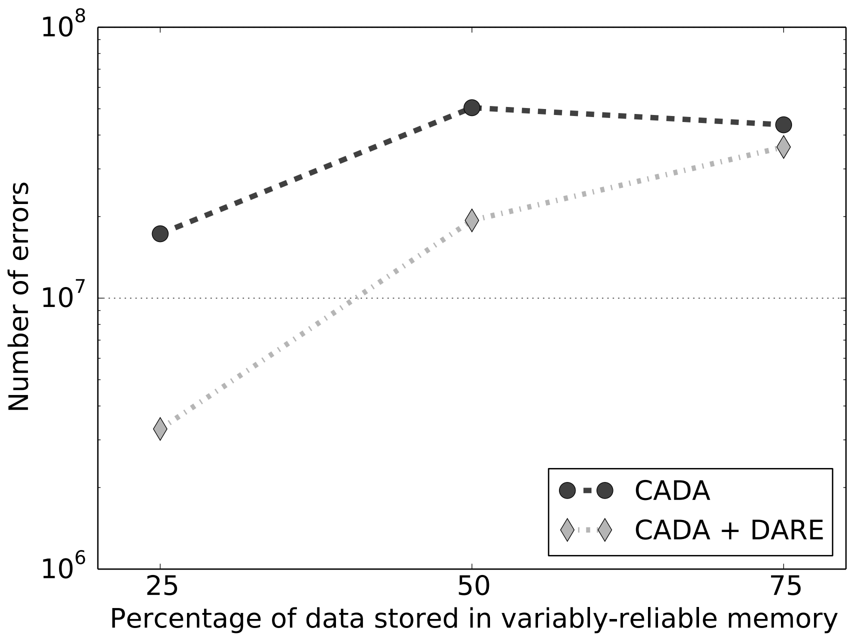
- Data classification based on criticality
  - ✓ E.g. low/high-frequency coefficients
- Refresh by access
  - ✓ Exploit the natural refresh
  - ✓ Spread accesses to variably-reliable memory
  - ✓ Iterative algorithms (e.g. k-means)
  - ✓ Controlled anti-locality techniques (e.g. stencils)
- Access-aware scheduling
  - ✓ Postpone writes to variably-reliable memory
  - ✓ Prioritize reads to variably-reliable memory

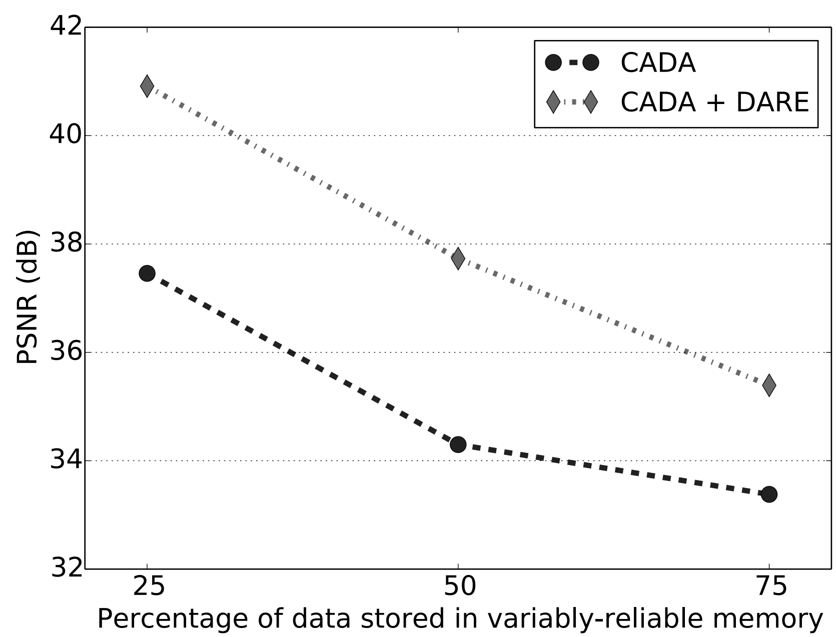
Refresh-by-data-access

- ✓ Accesses during window of vulnerability act as natural refresh
- ✓ Move writes late, move reads early
- ✓ Scheduling controls data refresh time
- ✓ Anti-locality optimization problem



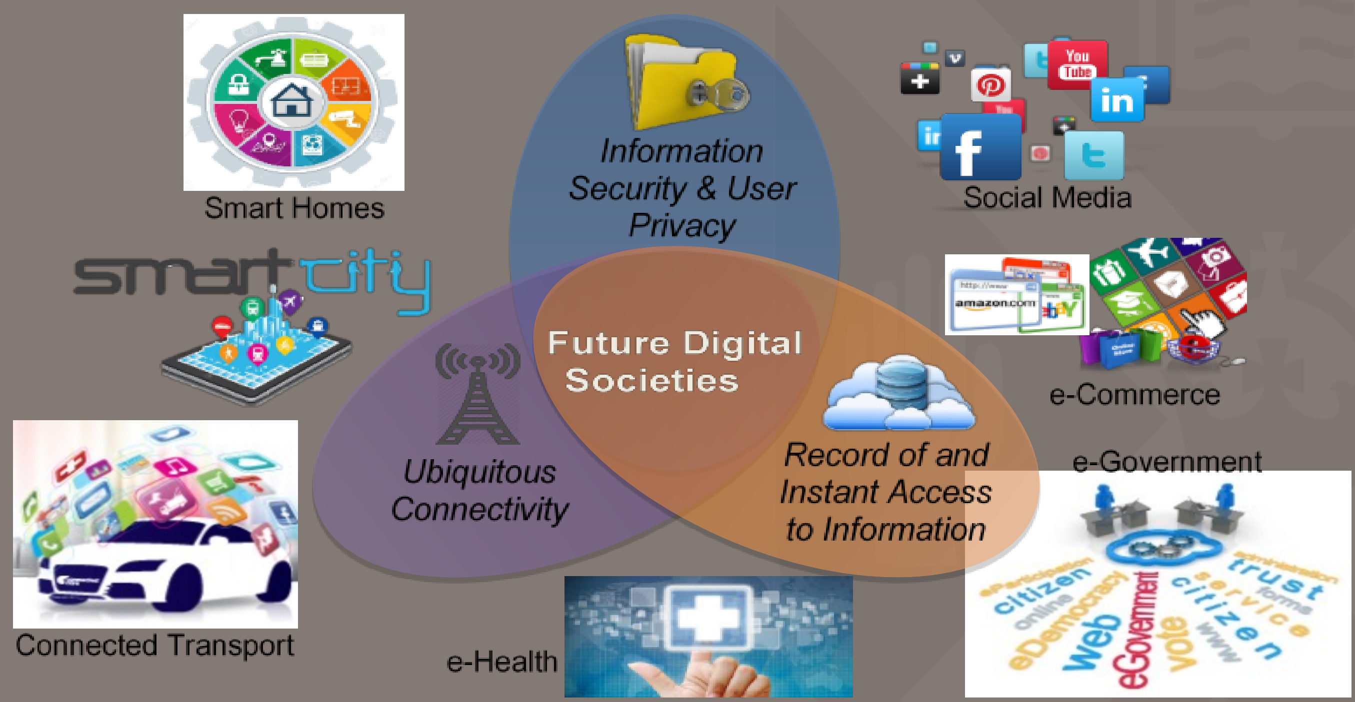
Refresh-by-access





WE ARE QUEEN'S UNIVERSITY BELFAST
We are exceptional

HPC in a different context



Queen's University Belfast

Acknowledgments

- The team
  - ✓ Charalampos Chalios
  - ✓ Kostas Tovletoglou
  - ✓ Giorgis Georgakoudis
  - ✓ George Karakonstantis
  - ✓ Hans Vandierendonck
- The support
  - ✓ EPSRC (SERT)
  - ✓ EU (SCoRPiO, UniServer)
  - ✓ Royal Society (Wolfson Award)