Energy-Efficient Stochastic Matrix Function Estimator for Graph Analytics on FPGA

Heiner Giefers, Peter Staar, Raphael Polig

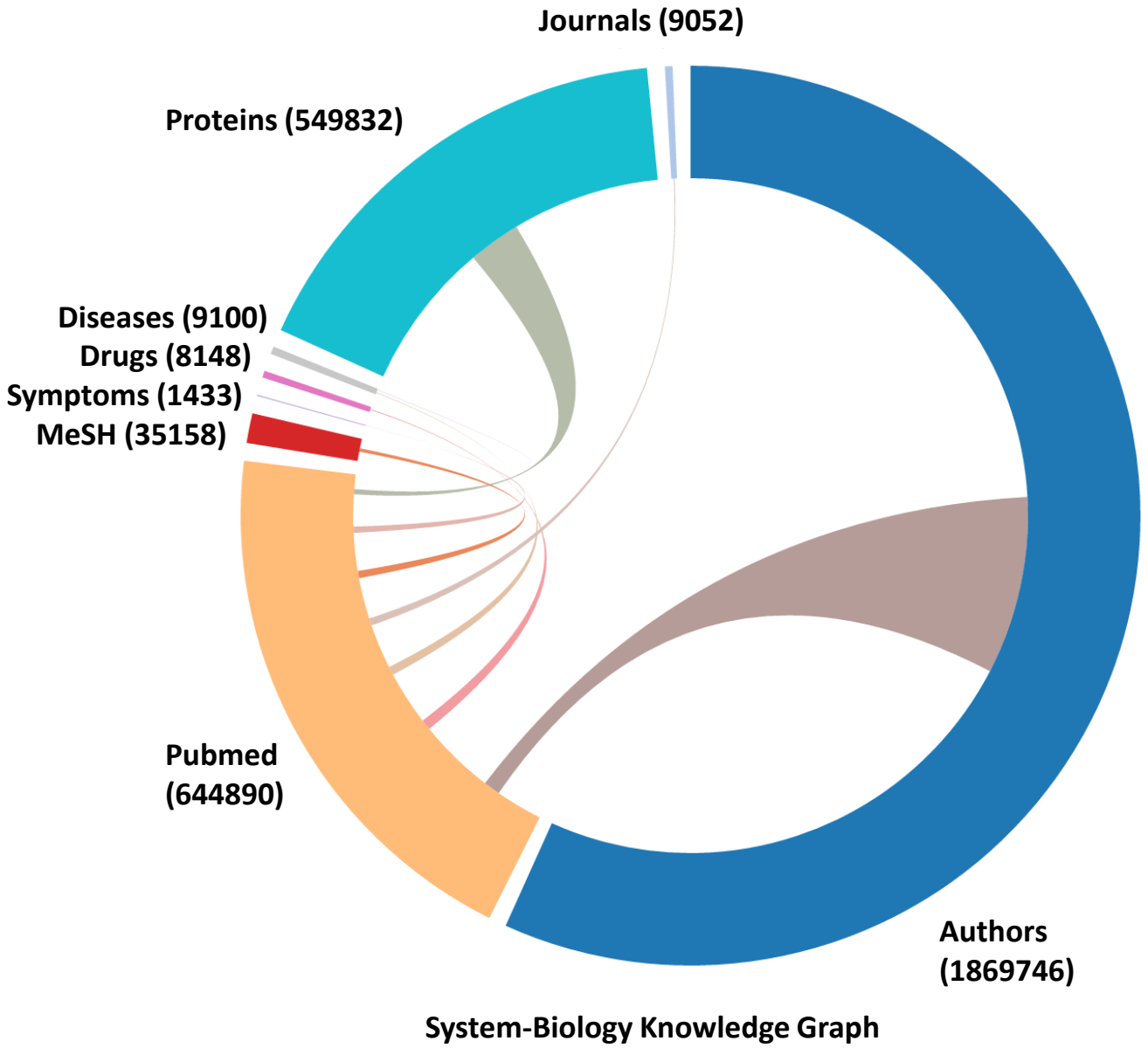
IBM Research – Zurich

26th International Conference on Field-Programmable Logic and Applications
29th August – 2nd September 2016
SwissTech Convention Centre
Lausanne, Switzerland

IBM Research

Motivation

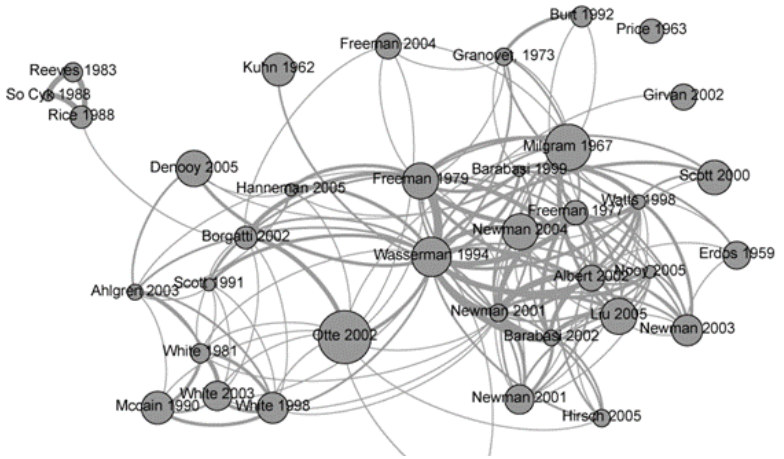
- Knowledge graphs appear in many areas of basic research
- These knowledge graphs can become very big (e.g. cover around ~80M papers and 10M patents)
- We want to extract hidden correlations in these graphs

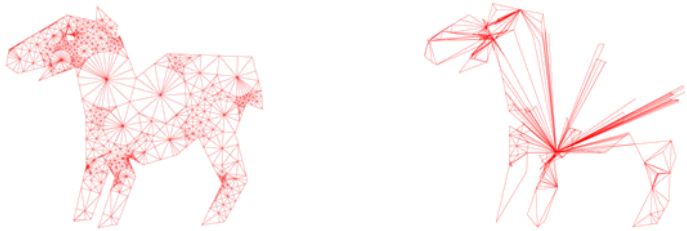


Graph Analytics Use Cases

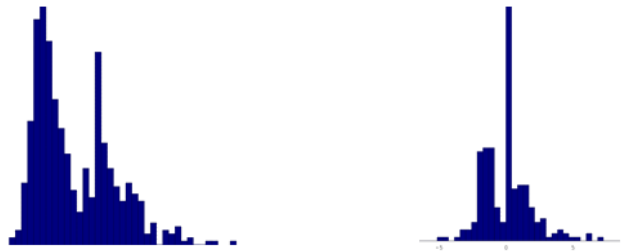
To extract hidden correlations in these graphs, we need to apply advanced graph-algorithms. Examples are:

1. Subgraph-centralities: Find the most relevant nodes by ranking them according to the number of closed wa





2. Spectral-methods: Compare large graphs by looking at their spectrum



IBM Research

Graph Analytics Use Cases

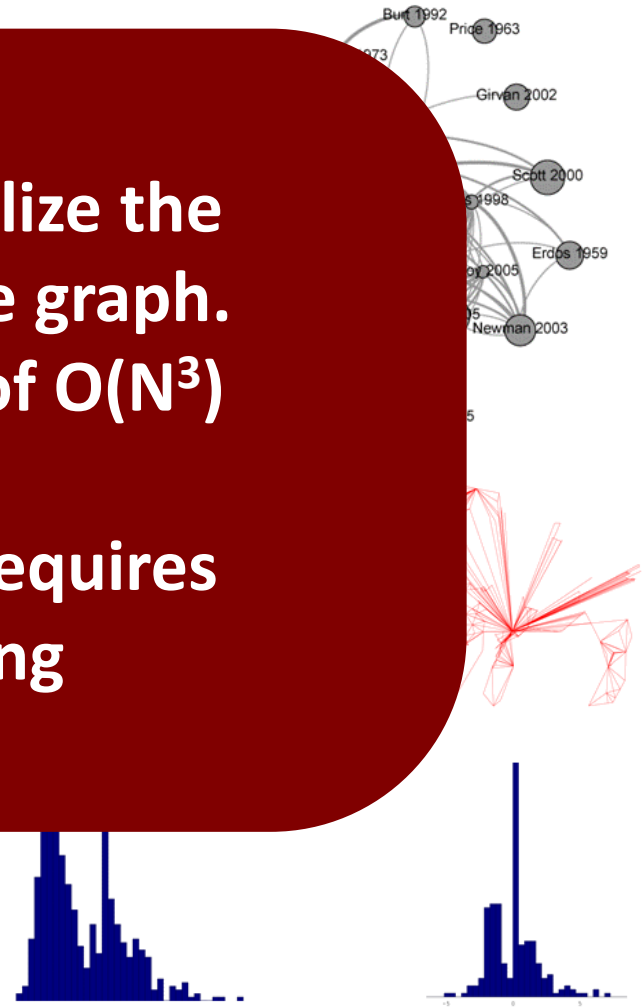
To extract hidden correlations in these graphs, we need to apply advanced graph-algorithms. Examples are:

1. Subgraph-ce
   relevant nod
   according to

2. Spectral-met
   graphs by lookin

Requires us to diagonalize the adjacency matrix of the graph. This has a complexity of $O(N^3)$

A graph of 1M nodes requires exascale computing



IBM Research

Node Centrality for Ranking Nodes in a Graph

- Subgraph centrality
  - Total number of closed walks in the network
  - The number of walks of length l in A from u to v is $\left(A^l\right)_{uv}$
  - Subgraph centrality considers all possible walks, shorter walks have higher importance:

    $1 + A + {A^2}/{2!} + {A^3}/{3!} + {A^4}/{4!} + {A^5}/{5!} + \cdots$
  - Taylor series for the exponential function e^A → weighted sum of all paths in A
  - Consider only closed walks → $c_i = (Diag[e^A])_i$

- Explicit computation of matrix exponentials is difficult
  - Though A is sparse, A^l becomes dense → huge memory footprint
  - Exascale compute requirements for exact solutions

Observations

- Observation 1: We only need an **approximate solution**
  - We do not need highly accurate results to obtain a good ranking!
  - We do not need to know exact value of the eigenvalues in order to have a histogram of the spectrum of A!

- Observation 2: In both operations, we need to compute a **subset of elements** of a matrix-functional
  - In the case of the subgraph-centrality, we need the diagonal of e^A
  - In the case of the spectrogram, we need to compute the trace of multiple step-functions

Stochastic Matrix-Function Estimator (SME)

Framework to approximate (a subset of elements of) the matrix *f*(A), where *f* is an arbitrary function and A is the adjacency matrix of the graph [1].

```
R = zero();
for l = 1 to Ns/Nb do
  forall e in V do
    e = (rand()/RAND_MAX<0.5) ? -1.0 : 1.0;
  done
  M0 = V
  W = c[0] * V                  // AXPY
  M1 = A * V                    // SPMM
  W = c[1] * M1 + W             // AXPY
  for m = 2 to Nc do
    M0 = 2 * A * M1 - M0        // SPMM
    W = c[m] * M0 + W           // AXPY
    pointer_swap(M0,M1)
  done
  R += W * V^T                  // SGEMM / DOT
done
E[f(A)] = R/Ns
```

Use Ns test vectors in blocks of size Nb

Initialize the Nb columns of V with random -1/1 **(2%)**

Compute W = f(A) V with Chebyshev polynomials of the first kind. **(97% of run time)**

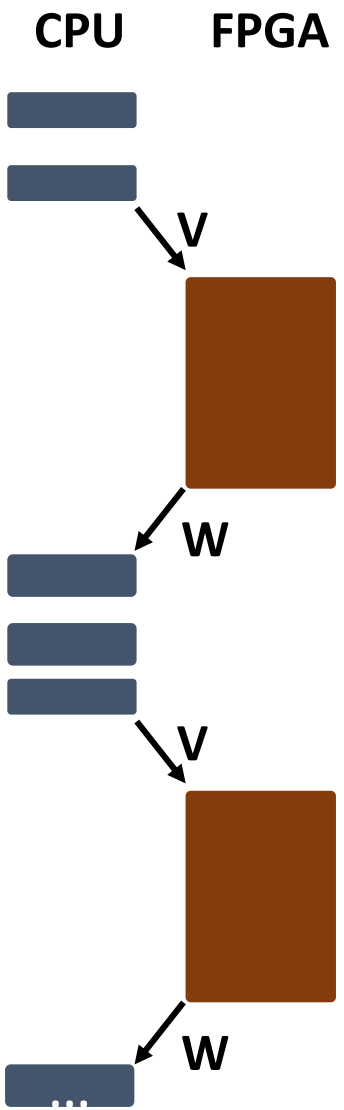
Accumulate partial results over test vectors **(1%)**

Normalize to get final result

[1] Peter W. J. Staar, Panagiotis Kl. Barkoutsos, Roxana Istrate, A. Cristiano I. Malossi, Ivano Tavernelli,Nikolaj Moll, Heiner Giefers, Christoph Hagleitner, Costas Bekas, and Alessandro Curioni. “Stochastic Matrix-Function Estimators: Scalable Big-Data Kernels with High Performance.” IPDPS 2016. (received Best Paper Award)

Accelerated Stochastic Matrix-Function Estimator

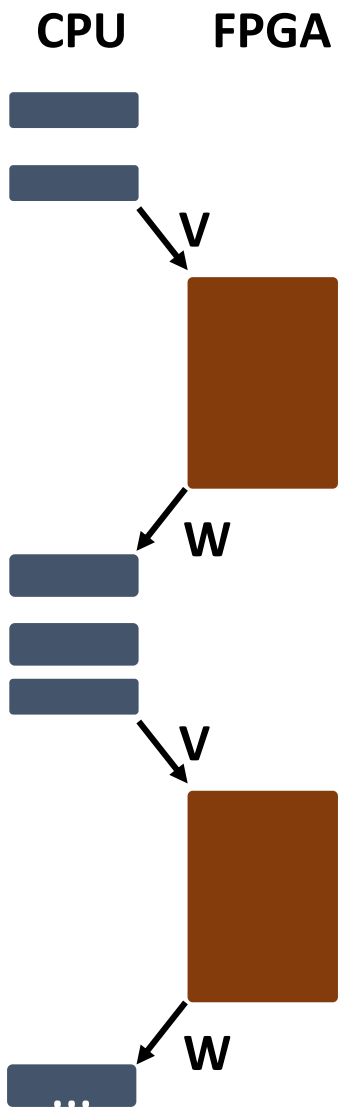
```
R = zero();
for l = 1 to Ns/Nb do
  forall e in V do
    e = (rand()/RAND_MAX<0.5) ? -1.0 : 1.0;
  done
  M0 = V
  W = c[0] * V                    // AXPY
  M1 = A * V                      // SPMM
  W = c[1] * M1 + W               // AXPY
  for m = 2 to Nc do
    M0 = 2 * A * M1 - M0          // SPMM
    W = c[m] * M0 + W             // AXPY
    pointer_swap(M0,M1)
  done
  R += W * V^T                    // SGEMM / DOT
done
E[f(A)] = R/Ns
```

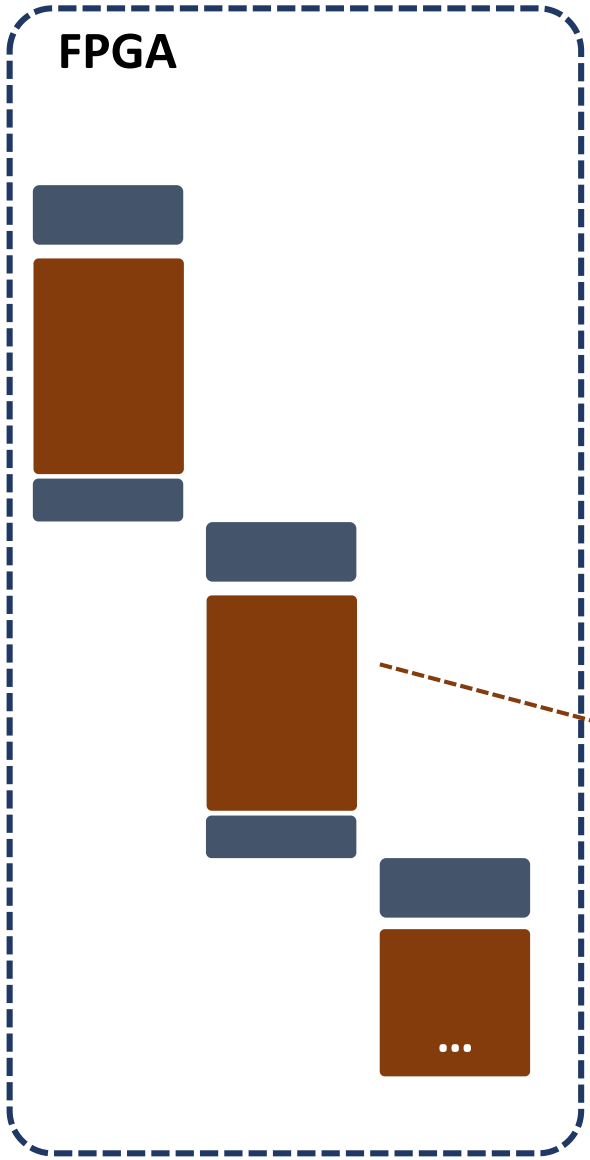


IBM Research

Accelerated Stochastic Matrix-Function Estimator

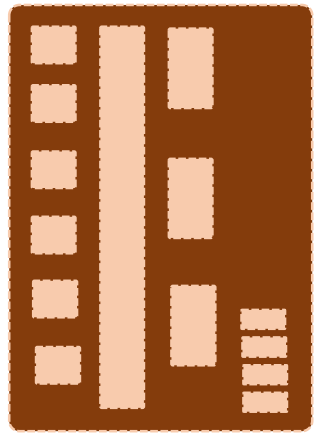
```
R = zero();
for l = 1 to Ns/Nb do
  forall e in V do
    e = (rand()/RAND_MAX<0.5) ? -1.0 : 1.0;
  done
  M0 = V
  W = c[0] * V                    // AXPY
  M1 = A * V                      // SPMM
  W = c[1] * M1 + W               // AXPY
  for m = 2 to Nc do
    M0 = 2 * A * M1 - M0          // SPMM
    W = c[m] * M0 + W             // AXPY
    pointer_swap(M0,M1)
  done
  R += W * V^T                    // SGEMM / DOT
done
E[f(A)] = R/Ns
```





Map the entire outer loop onto the FPGA

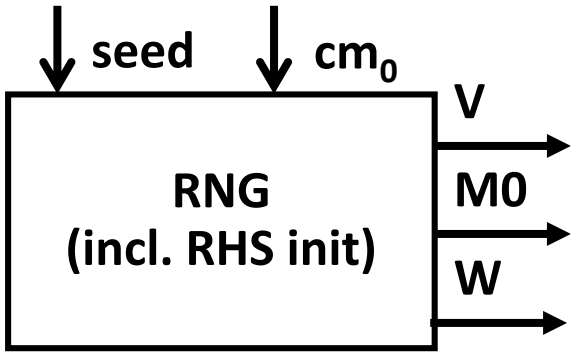
- (Almost) no host-device communication
- 3 sequential stages
- No double buffering needed
- 4 asynchronous kernels in inner loop



IBM Research

SME Architecture – Random Number Generator

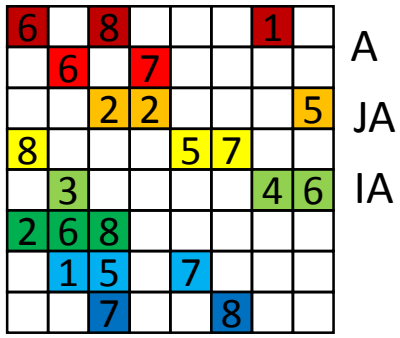
- xorshift64 based random number generator to generate Rademacher distribution
  - High quality, passes many passes many statistical tests [2]
  - Well suited for FPGA implementation
  - Initialize V, M0, and W on-the-fly



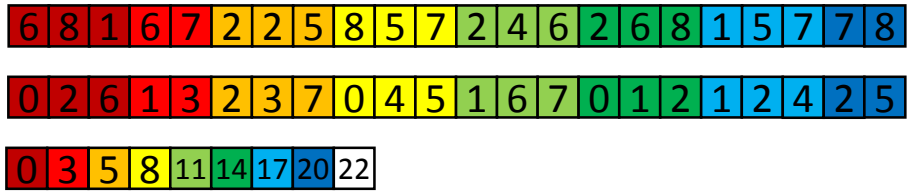

```
ulong2 xorshift64s (ulong x){
          ulong2 res;
           x ^= x >> 12;
           x ^= x << 25;
           x ^= x >> 27;
          res.x = x;
          res.y = x * 2685821657736338717ull;
          return res;
}
__kernel
void rng(float *M0,*W,*V,cm, uint num, ulong seed){
  ulong2 rngs = {rand, 0xdecafbad};
  ulong rs; float rn;
  for(unsigned k = 0; k < num; k+=N_UNROLL){
    rngs = xorshift64s(rngs.x);
    rs = rngs.y;
    #pragma unroll N_UNROLL
    for(unsigned b = 0; b < N_UNROLL; b++){
      rn = ((rs >> b) & 0x1) ? -1.0 : 1.0;
      V[k+b] = rn;
      M0[k+b] = rn;
      W[k+b]  = cm*rn;
    }
}
```

[2] George Marsaglia. “Xorshift RNGs,” Journal of Statistical Software, 2003.

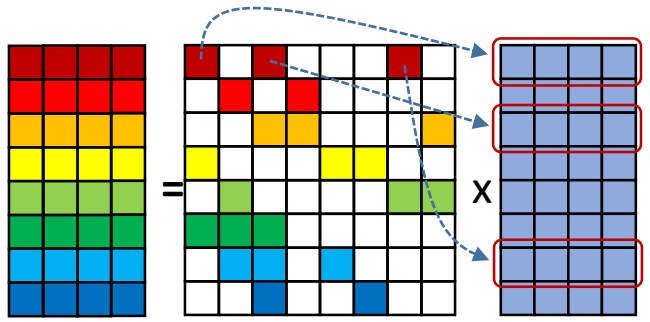
SME Architecture: CSR Sparse Matrix Multiplication



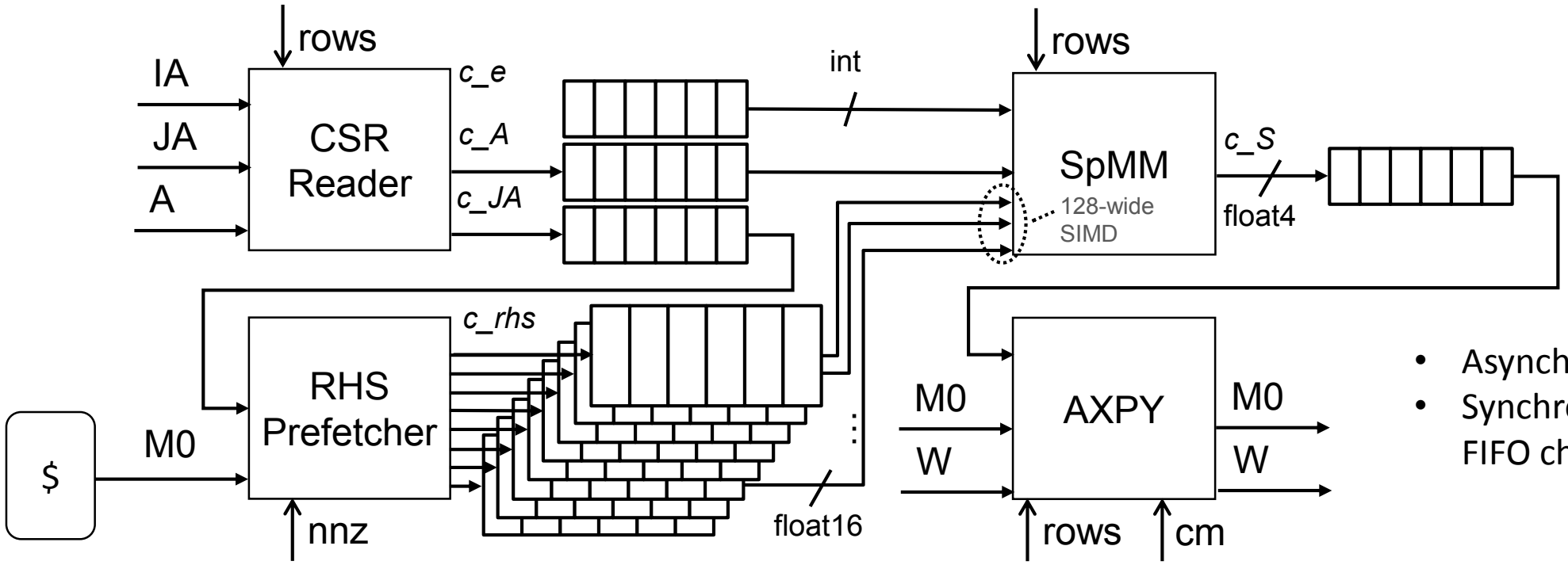
A
JA
IA



sparse matrix in CSR format



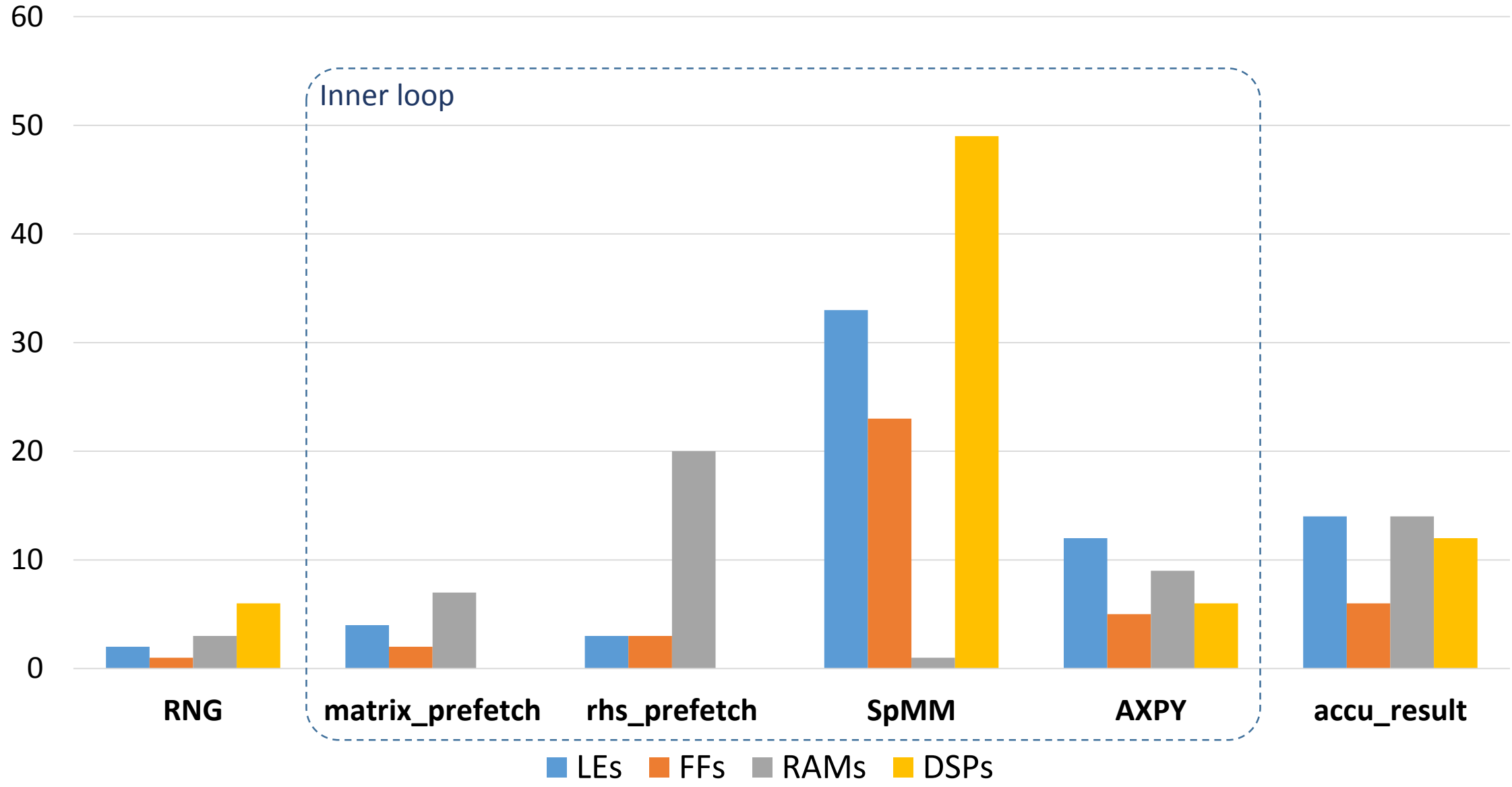
sparse matrix-matrix multiplication



- Asynchronous kernels
- Synchronization via FIFO channels

IBM Research

Resource Utilization for Kernels on Stratix-V 5SGXA7



IBM Research

SME on Heterogeneous System

POWER8 heterogeneous node

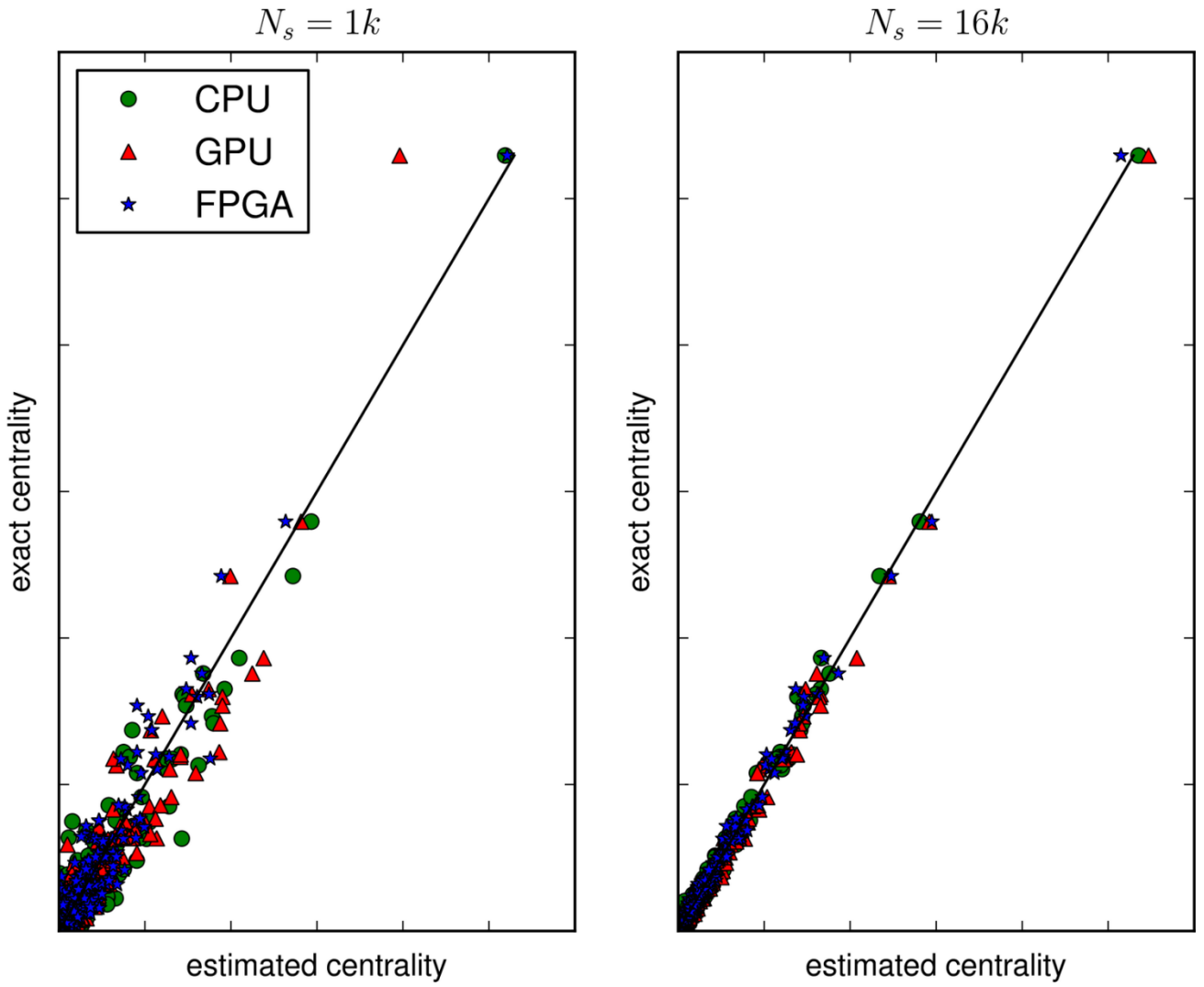
1. Dual-socket 6-core CPU, 96 threads
   - IBM xlC compiler using OpenMP and Atlas BLAS

2. NVIDIA Tesla K40 GPU
   - CUDA 7.5 with cuBLAS
   - Self-developed SpMM outperforms cusparseScsrmm()

3. Nallatech PCIe-385 card w/ Altera Stratix-V FPGA
   - Altera OpenCL HLS

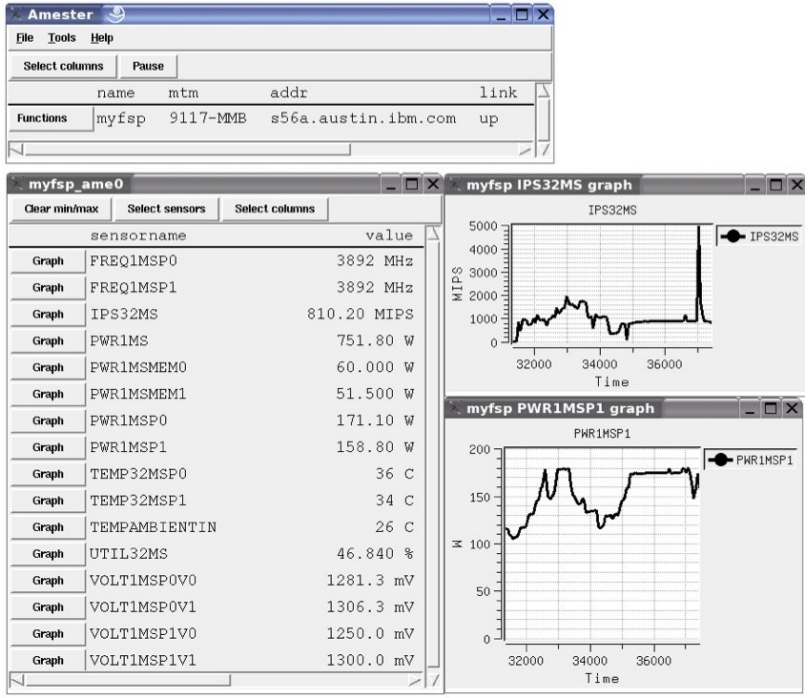
SME – Approximation Quality on the 3 Platforms

- Estimation quality depends on several factors
  - Number of test vectors
  - Number of terms in Chebyshev expansion
  - Quality of the random number generator used to initialize the test vectors
  - Precision of floating point operations



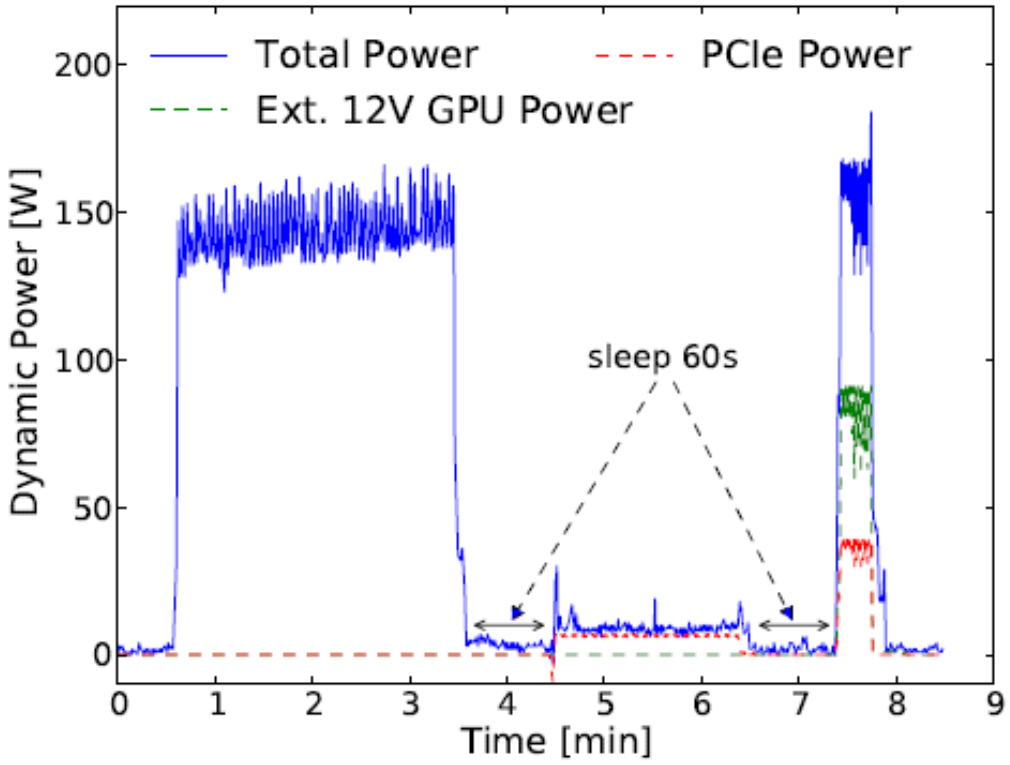
Power Profiling

- POWER8 On-Chip Controller (OCC)
  - Enables fast, scalable monitoring (ns timescale)
  - OCC is implemented in a POWERPC 405
  - Uses continuous running, real-time OS
  - Monitors workload activity, chip temperature and current

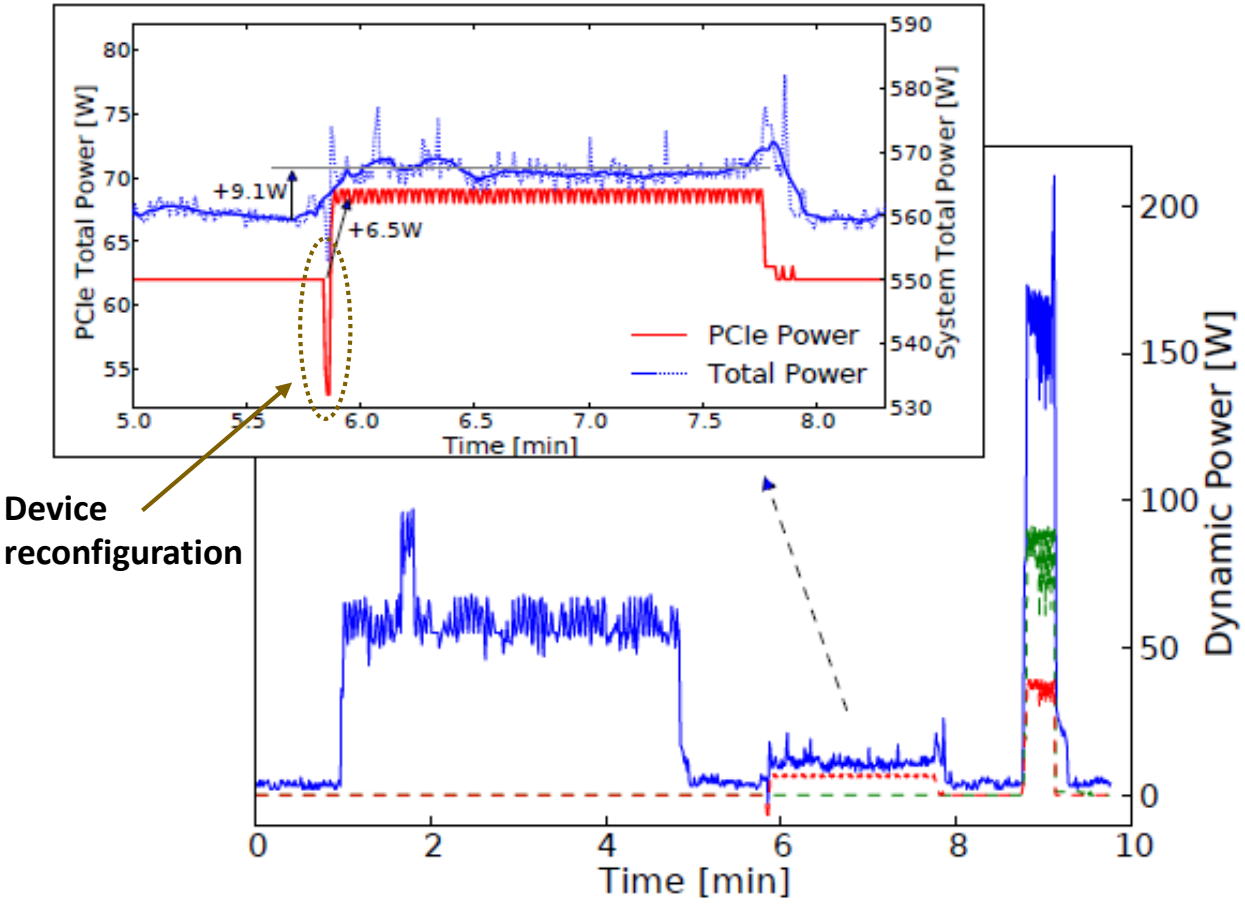


- Trace power consumption using Amester
  - Tool for out-of-band monitoring of POWER8 servers
  - Open sourced on github: **`github.com/open-power/amester`**
  - Current sensors for various domains (socket, memory buffer/DIMM, GPU, PCIe, fan, …)
  - Compute power consumption: $P^{comp} = P^{total} - P^{idle}$

Application-Level Power Traces



CPU (6 threads) FPGA GPU



CPU (1 thread)

IBM Research

SME – Energy-Efficiency Analysis

Platform	Run time [s]	Dynamic Power [W]	Energy to Solution [kJ]
CPU	172.55	143.92	24.83
CPU	232.31	57.01	13.24
GPU	19.52	155.42	3.03
FPGA	114.00	9.13	1.04

Fastest CPU version (6 threads)

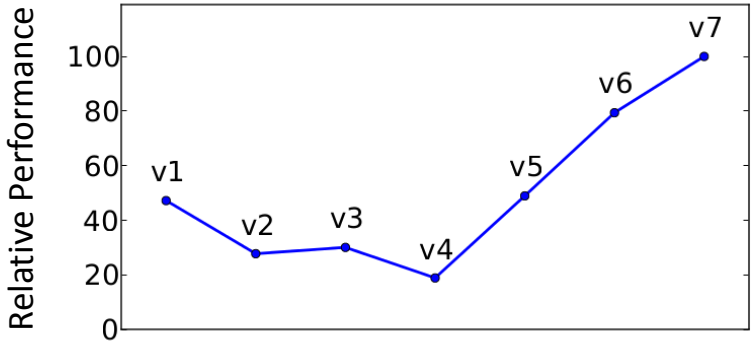
Most efficient CPU version (1 thread)

FPGA is ~6x slower but ~3x more energy-efficient compared to the GPU

CPU	IBM POWER8 2-socket 12-core
FPGA	Nallatech PCIe-385 with Altera Stratix-V
GPU	NVIDIA K40

Conclusion

- Accelerators outperform the CPU. GPUs are dominant in terms of absolute performance
  - GPU is 12x, FPGA 2x faster than a CPU core
- The compute energy for the FPGA outstanding
  - 3x better compared tor GPU, 13x better compared to the CPU
- What about the idle power? (~550W for the system we used)
  - We need energy-proportional computing
  - Cloud: Accelerators free CPU cycles
  - Cloud-FPGA: Standalone, network-attached FPGA to remove “host overhead”
- OpenCL increased productivity
  - Short design time, (almost) no verification
  - Optimization is cumbersome



Questions?

Heiner Giefers

IBM Research – Zurich

hgi@zurich.ibm.com

26th International Conference on Field-Programmable Logic and Applications
29th August – 2nd September 2016
SwissTech Convention Centre
Lausanne, Switzerland

IBM Research