Computer Arithmetic in High Performance Reconfigurable Computing

George A. Constantinides[1]

Imperial College London

January 15, 2012

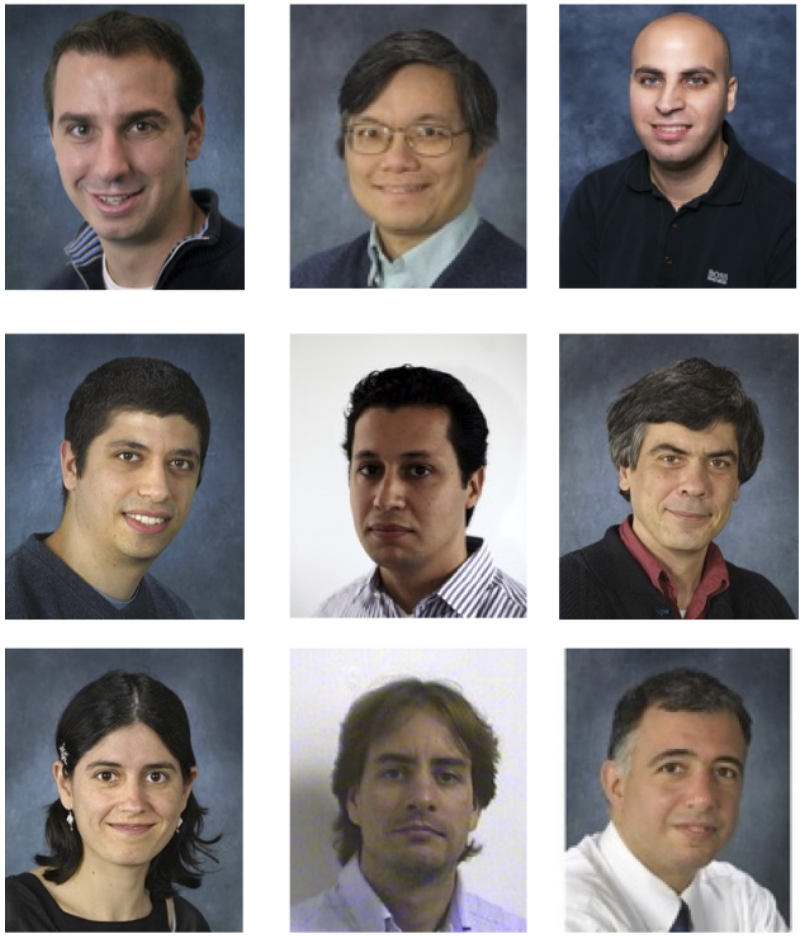
[1]Thanks to: Mr Juan Jerez

Outline

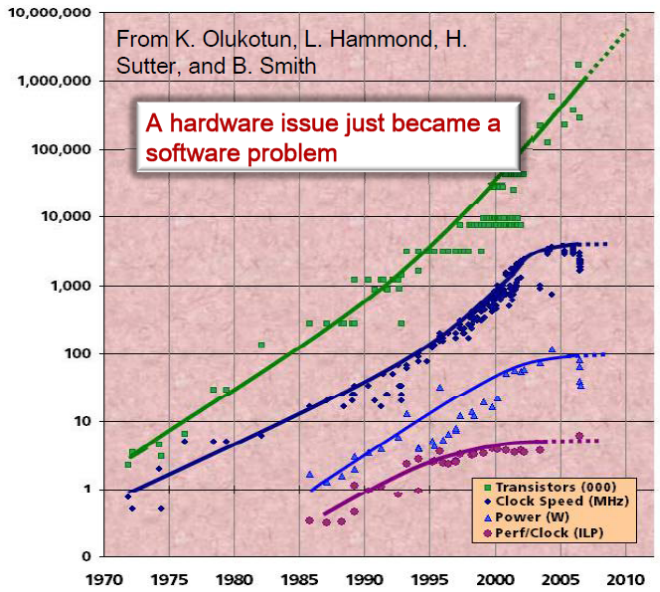
- Returning silicon to computation: some options, and convergence
- Reconfigurable computing: the FPGA
- Compilation driven by accuracy specifications: opening windows
- Computer arithmetic: numerical and hardware considerations
- A case study: accelerating the Lanczos algorithm (for LE solution)

Our Research Group

- Circuits and Systems Group
- One of 5 research groups in EEE at Imperial
- In numbers:
  - 9 academic staff
  - c.20 research staff
  - c.40 PhD students
- Analogue electronics: medical devices
- Digital electronics: programmable devices, tools and applications
- My support from EPSRC, EC, Altera, Xilinx, ARM, Imagination, ESA and Mathworks.

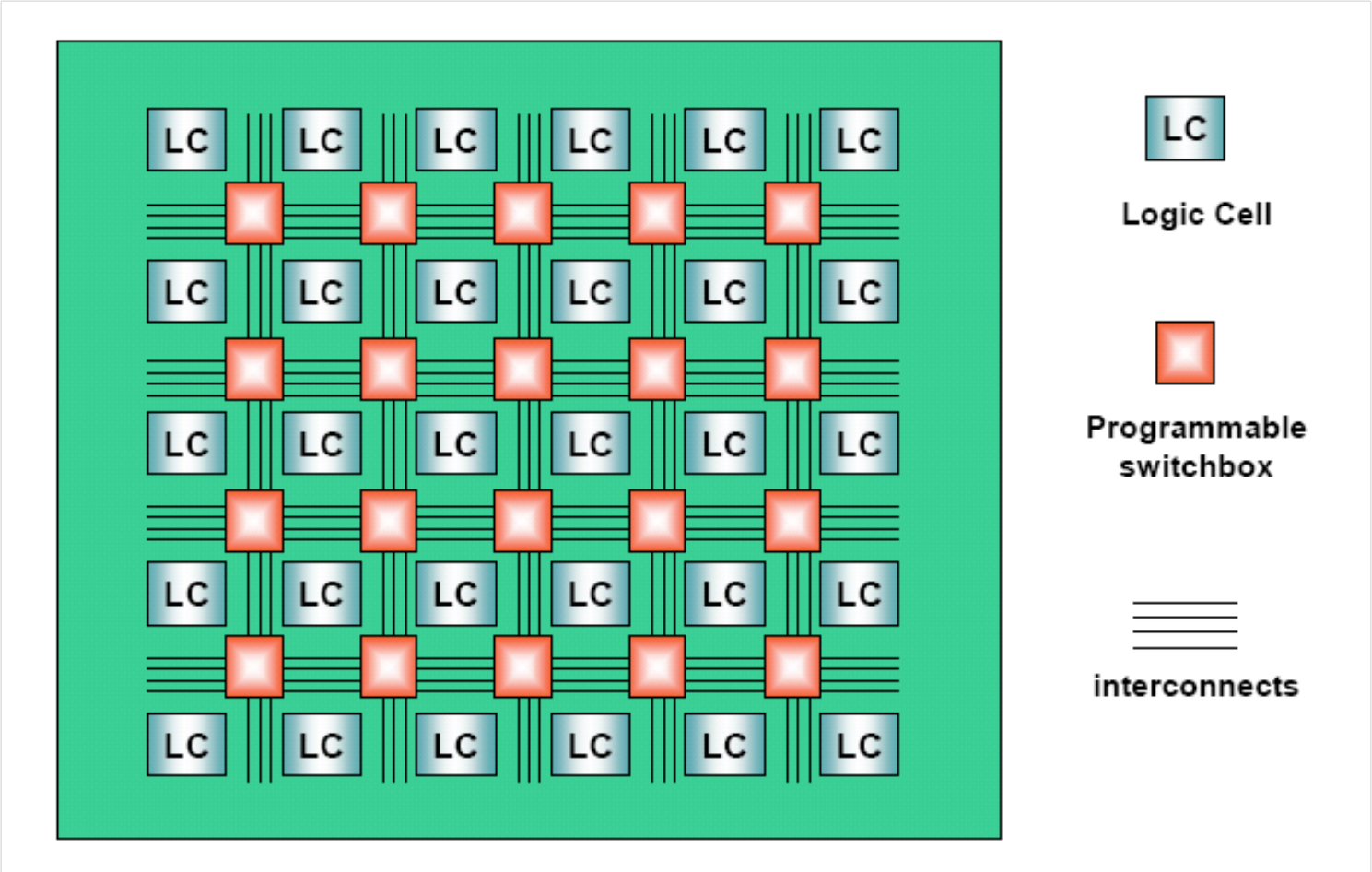


Computer Architecture is Changing



- All about silicon efficiency! Cache or computation?
- Fixed-functionality hardware too expensive - what's the *right* way to introduce programmability?
- GPU: Hundreds of cores. Explicit memory management. Singles not doubles. Simple task parallelism.
- FPGA: Tens of thousands of 'cores'. Explicit memory management. Right-sized datapath. Arbitrary structure.

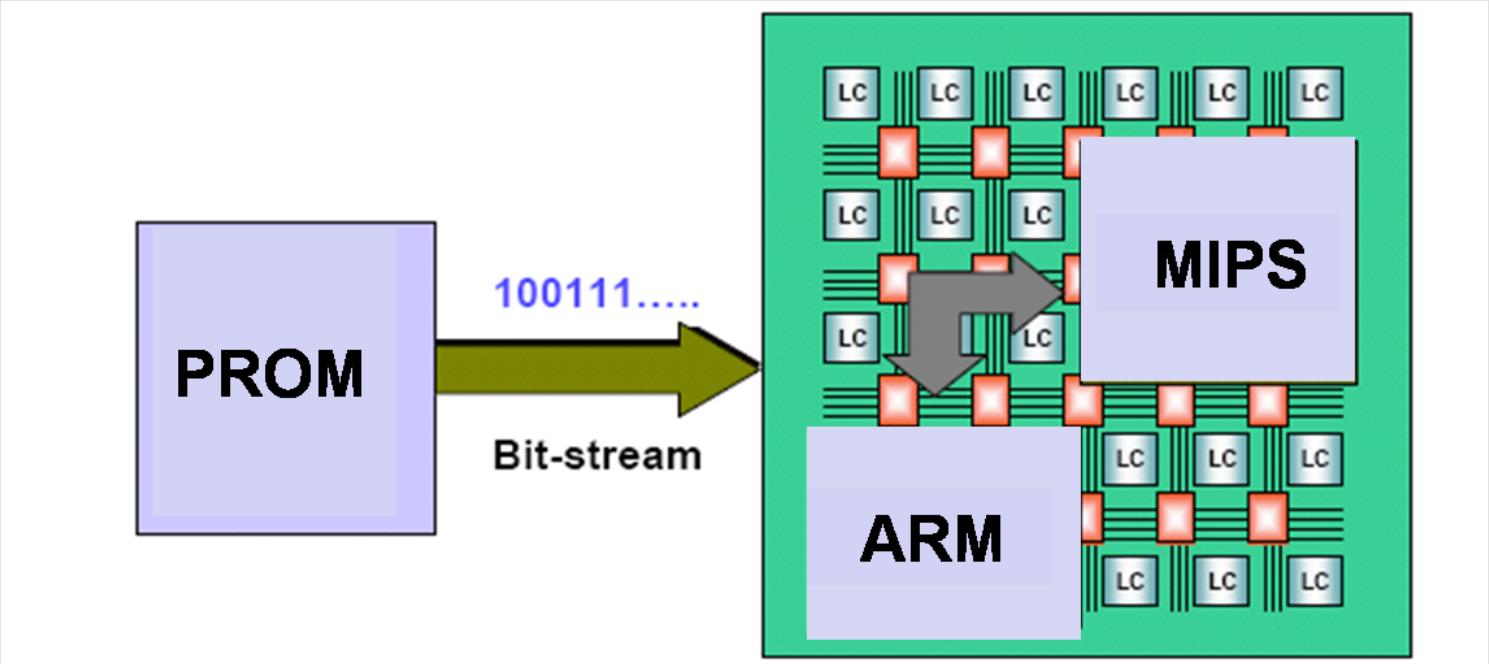
What is a Field-Programmable Gate Array?



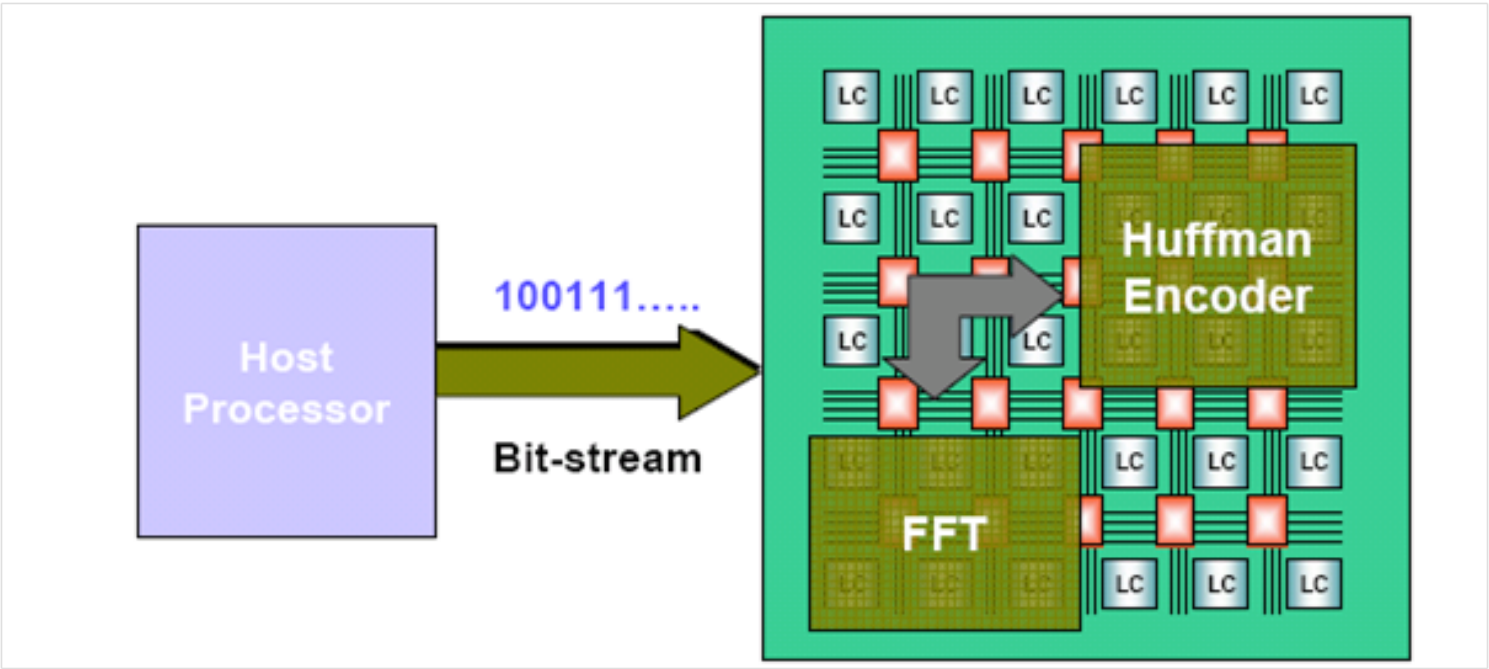
- Regular stucture, so at technology leading edge.
- Convergence of architectures?

The FPGA as a GPP Emulator

Recently popularised by Berkeley: 'quick' emulation of manycore architectures.



The FPGA as a GPP!

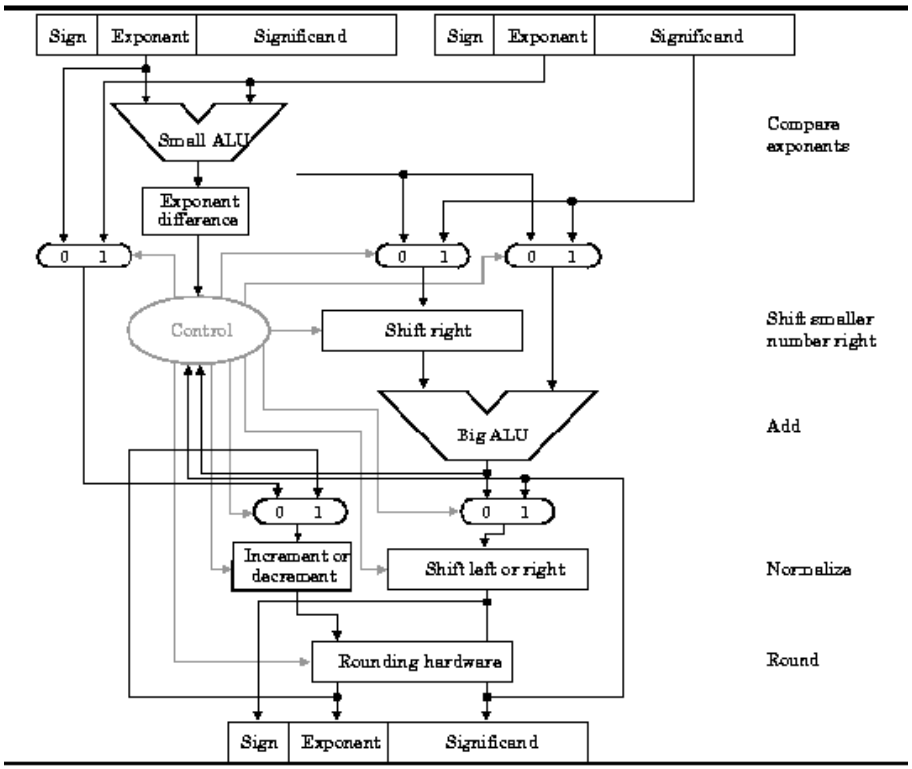


- Select an architecture for your algorithm.
- High speed (*e.g.* 20x LP), low power (*e.g.* 8x MC).

Representing the Reals

- Hardware represents numerical data as bit strings.
- A bit string of length n can represent at most 2^n distinct values.
- Representations vary in how these strings are mapped onto reals: $f : \{0,1\}^n \rightarrow \mathbb{R}$.
  - Floating-point (s-m-e),
  - Fixed-point (s-m,2c,1c),
  - LNS (s-e),
  - RNS (m,m,...), *etc.*
- We have always cared about using 'enough' precision. Now we should care about using 'just enough' precision.
  - Lower precision $\Rightarrow$ smaller area units, less bandwidth $\Rightarrow$ more units, more transfer $\Rightarrow$ faster.
  - Remember: It's all about silicon efficiency!

Hardware Implications



(from http://www.cise.ufl.edu/~mssz/CompOrg/CDA-arith.html)

- Floating-point (m,e) bits:
  - $\Theta(m+e)$ bandwidth
  - $\Theta(m^2+e)$ adder (ripple-carry, barrel shift)
  - $\Theta(m^2+e)$ multiplier (array/Wallace/Dadda, ripple-carry)
- Fixed-point n bits:
  - $\Theta(n)$ bandwidth
  - $\Theta(n)$ adder (ripple-carry)
  - $\Theta(n^2)$ multiplier (array/Wallace/Dadda)

Addition: Fixed vs Float

Table: Resource utilization and latency for a single adder on a Virtex7 XT 1140 using Xilinx Core Generator floating point v5.0

	Registers	LUTs	Latency
`double`	1046	911	14
`float`	557	477	11
`FX53`	53	53	1
`FX24`	24	24	1

- $\sim 20x$ resource savings
- $\sim 10x$ latency savings

Our Vision

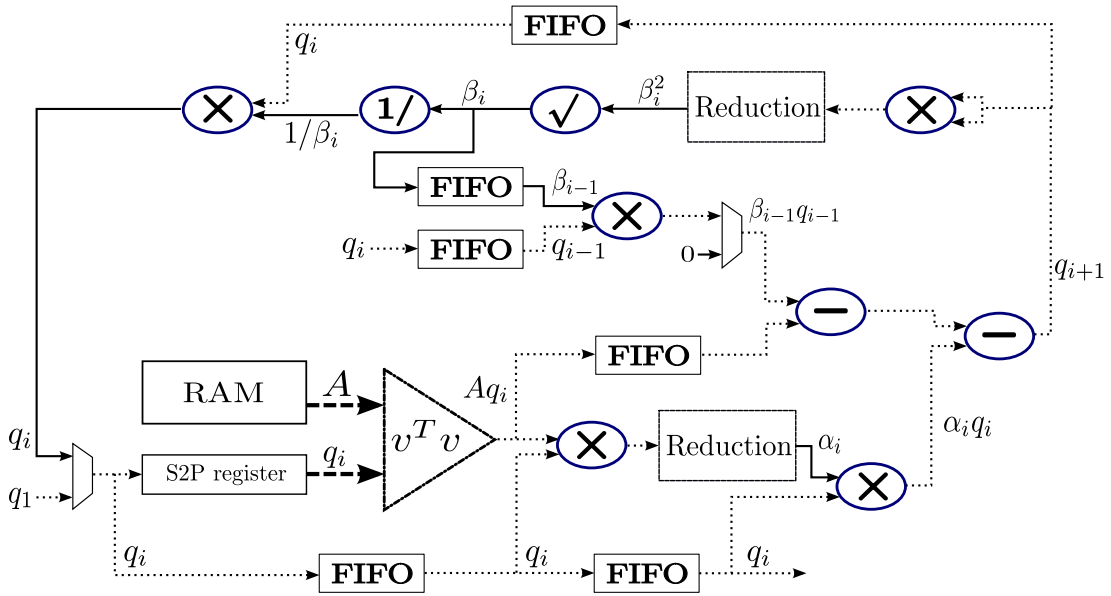
The Central Problem

$$\max_{s,p} \text{perf}(s,p)$$
$$\text{subject to} : \forall i.\text{Pre}(i) \rightarrow \text{Out}(s,p,i) \in \text{Accept}(i). \quad (1)$$

- Here s denotes circuit structural parameters, and p denotes circuit precisions. Note plural - we are in a parallel environment, so we should take advantage!
- Select degree of parallelism, circuit structure, number representation(s) and precisions to maximise performance on a given architecture.

The Lanczos Algorithm

Given q_1 such that $\|q_1\|_2 = 1$
and an initial value $\beta_0 := 1$
for $i = 1$ to ... i_{max} **do**
$q_i \leftarrow \frac{q_i}{\beta_{i-1}}$
$z_i \leftarrow Aq_i$
$\alpha_i \leftarrow q_i^T z_i$
$q_{i+1} \leftarrow z_i - \alpha q_i - \beta_{i-1} q_{i-1}$
$\beta_i \leftarrow \|q_{i+1}\|_2$
end for



Latency (cycles) given by

$$\lceil \frac{N}{P} \rceil + l_A * \lceil log_2(N) \rceil + 5l_M + l_A + l_{SQ} + l_D + 2 + 2l_{red}$$

$$l_{red} := \lceil \frac{N}{P} \rceil + l_A * \lceil log_2(P) \rceil + l_A + l_A * \lceil log_2(l_A) \rceil - 1$$

Fixed point? Peak bounds required on...

- q_i
- A
- Aq_i
- α_i
- $\beta_i q_{i-1}$
- $Aq_i - \beta_{i-1} q_{i-1}$
- $\alpha_i q_i$
- q_{i+1}
- $q_{i+1}^T q_{i+1}$
- β_i
- $\frac{1}{\beta_i}$

Preconditioning

Instead of solving

$$Ax = b$$

solve

$$M^{-\frac{1}{2}} A M^{-\frac{1}{2}} y = M^{-\frac{1}{2}} b \tag{2}$$

$$\widehat{A} y = \widehat{b} \tag{3}$$

The solution to the original problem can be recovered via

$$x := M^{-\frac{1}{2}} y$$

We propose a positive diagonal preconditioner whose elements are the absolute sum of the elements in each row of A, i.e.

$$M_{kk} = \sum_{j=1}^{N} |A_{kj}|,$$

Theorem

Given preconditioner M, the symmetric Lanczos algorithm applied to $\widehat{A} := M^{-\frac{1}{2}} A M^{-\frac{1}{2}}$, for any symmetric matrix A, has intermediate variables with the following bounds:

- $q_i \in (-1, 1)$
- $\widehat{A} \in (-1, 1)$
- $\widehat{A} q_i \in (-1, 1)$
- $\alpha_i \in (-1, 1)$
- $\beta_i q_{i-1} \in (-1, 1)$
- $\alpha_i q_i \in (-1, 1)$
- $\widehat{A} q_i - \beta_{i-1} q_{i-1} \in (-2, 2)$
- $q_{i+1} \in (-1, 1)$
- $q_{i+1}^T q_{i+1} \in (-1, 1)$
- $\beta_i \in (\epsilon, 1)$
- $\frac{1}{\beta_i} \in (0, 1/\epsilon)$

where ϵ is determined by termination condition.

Resource Constrained Framework

$$\min_{P,k} \text{ latency}$$

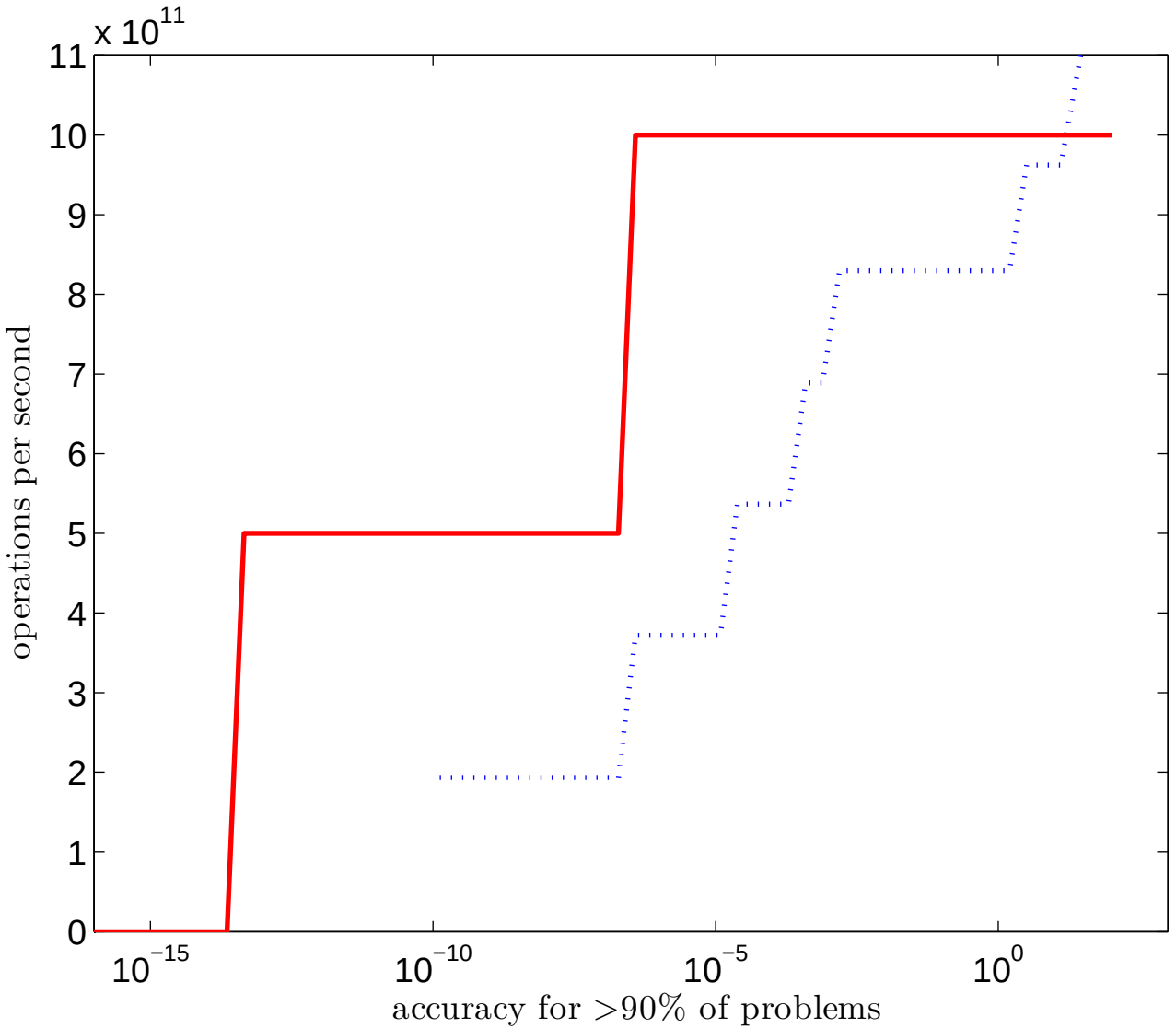
subject to

$$\mathbb{P}(e_k \leq \eta) > 90\% \tag{4}$$

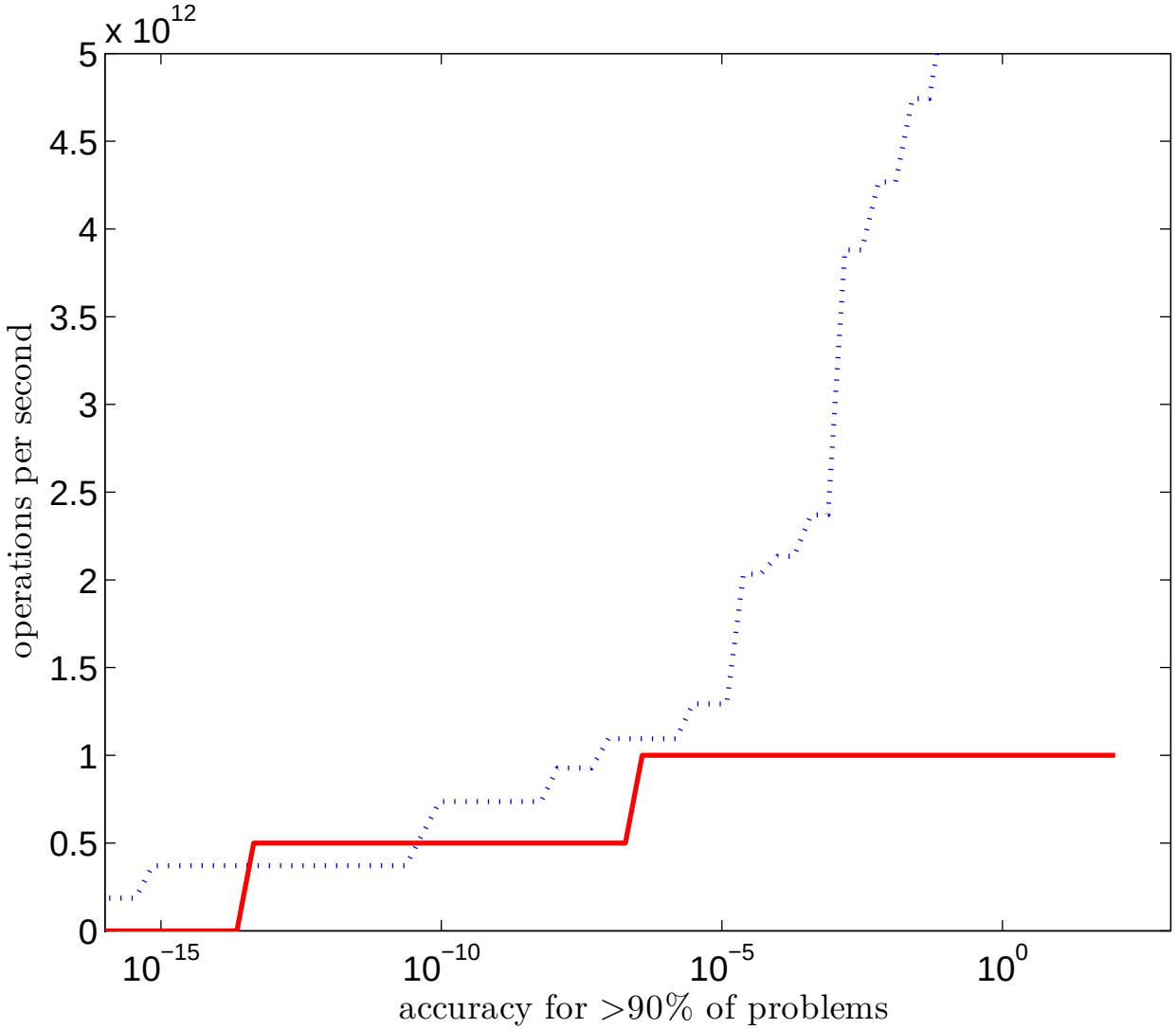
$$R(P,k) \leq \text{FPGA}_{\text{area}} \tag{5}$$

1. Find minimum number of bits k such that (4) is satisfied
2. Find maximum parallelism P such that (5) remains satisfied

Performance (N=750)



Performance (N=229) - Many problems



Conclusion

- Architecture is not what it was.
- Microelectronics design, NA, HPC, programming language theory, all have strong contributing roles to play.
- Need to devise suitable specification format, and compilers, for heterogeneous architectures *and look at the implications for those architectures*.
- Let's talk!

Some Adverts...