

A simulated annealing approach to task allocation on FPGAs

Philippa Conmy



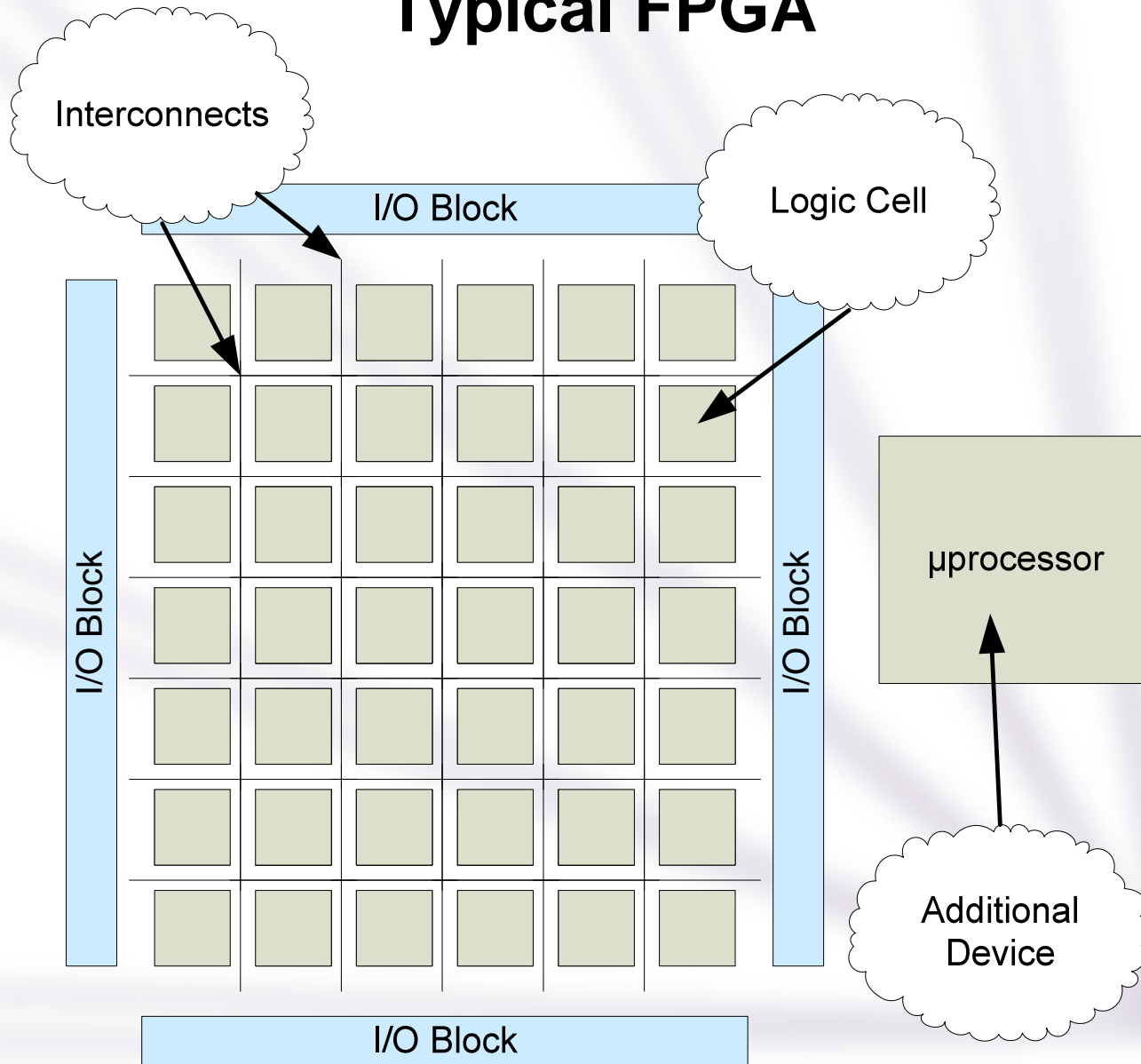
Talk Overview

- Introduction to problem
 - Description of approach
 - Description of fitness function
 - Results
 - Further work...
- Please ask questions as it's work in progress!

FPGAs

- Field Programmable Gate Arrays
 - 100,000s of logic gates which can be configured and reconfigured
 - Tasks run in parallel
 - Highly predictable
- Not always exploited to full potential for high integrity systems
 - Concern about bit flips and single event upsets (SEUs)
 - Designs can be overly conservative
 - Many empty cells
 - Lots of physical redundancy
 - External monitors

Typical FPGA



Improving Exploitation

- Can run multiple independent tasks
 - As tasks run in parallel there's limited resource contention
 - Can separate at interconnects
 - Multiple failures required before interference
 - Use separate I/O ports
- Advantage is that we can use fewer FPGAs
 - Better for weight, costs, power requirements
 - E.g. for small systems, UAVs
- How far can we take it?
- How do we manage multiple permutations?

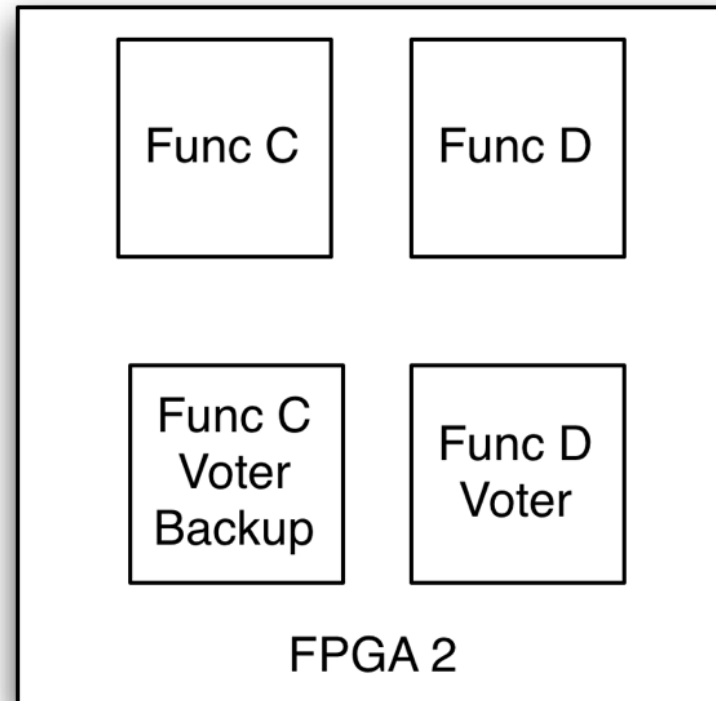
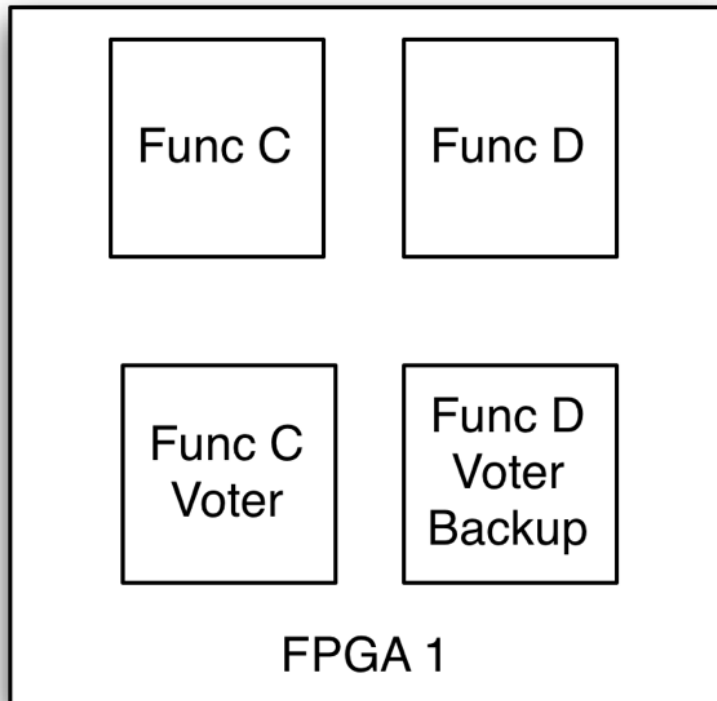
Reliability

- Many different aspects to safety critical systems
 - Availability, reliability, integrity etc.
- Focussed so far on reliability
 - I.e. ability to continuously provide a service
 - Standards such as IEC 61508 provide probability of failure targets for this
- Hope to expand the work at some point to consider other dependability aspects

Redundancy

- To improve the tolerance to failure we can
 - Replicate components
 - Mitigates against hardware failures
 - Use diverse components
 - E.g. diverse logic, diverse hardware
- For this work looked at N-Modular Redundancy
 - By replicating logic tasks can reduce the likelihood of loss of service due to physical failure
 - Either internal to the task or due to total FPGA failure
 - N.B. very early on discovered single voter system would not meet targets for highest integrity apps

Example



Method

- Use Simulated Annealing (SA) algorithm to develop solutions
 - Simple starting systems with single tasks
 - At each step along the way can either:
 - Add a copy of a task (2/19)
 - Add a new (empty) FPGA (6/19)
 - Move a task to different FPGA (9/19)
 - Delete an empty FPGA (2/19)
- Early experiments indicated deletion of FPGAs necessary
- Add dual voters on separate FPGAs
- Weightings in favour of adding and moving tasks

Method (cont)

- Overall aim
 - Minimise resource usage whilst ensuring reliability targets are met
 - Both measured for fitness at each stage
 - Want to provide configuration tool – on or offline
- At each stage calculate fitness of solution
 - If solution is better than previous solution keep it
 - If it's worse allow three extra steps before rolling back to previous best solution
 - Avoid getting trapped in local optimum
- FPGAs modelled as set of cells
 - Tasks require fixed set of cells

Resource Fitness – per FPGA

- Normalised value based on optimum usage of FPGA device

Category	Penalty	Criteria - %age of cells used on FPGA
Maximum	100	Less than 15% and over 80%
Medium	50	Between 15 and 45
Low	25	Over 45 and up to 60
Minimum	0	Over 60 and up to 80

Reliability Fitness

- Based on ability of system to meet reliability targets
 - Probability of failure per X number of hours
- System is considered to have failed when:
 - A majority of lanes are lost
 - Both voters are lost
- Need to consider
 - Probability of internal failure
 - Take overall figure for likelihood of HW failure e.g. for SEU
 - Size of task relates to likelihood of that failure affecting it
 - Probability of FPGA failure

Reliability Fitness

- Some issues with these calculations
- Overly Optimistic
 - Assuming that all tasks really are independent
 - Both physically and in terms of function
 - Assuming loss of FPGAs is totally independent of one another
- Overly Pessimistic
 - Not all failure will lead to incorrect output from a task
 - E.g. SEU in a LUT may or may not cause incorrect output
- Area for improvement!

Reliability Fitness – per system

Category	Penalty	Criticality	Criteria – closeness to required probability
Maximum	100	Catastrophic/ Hazardous	More than 5 times the probability target
		Major	More than 10 times the probability target
		Minor/none	More than 15 times the probability target
Large	80	Minor/none	0.1 times under the probability target
		Catastrophic/ Hazardous	More than 1.1 times the probability target
		Major	More than 5 times the probability target
		Minor/none	More than 10 times the probability target
Medium	40	Major	0.005 times under the probability target
		Catastrophic/ Hazardous	More than 1.01 times the probability target
		Major	More than 1.5 times the probability target
		Minor/none	More than 5 times the probability target
		Catastrophic/ Hazardous	0.0005 times under the probability target



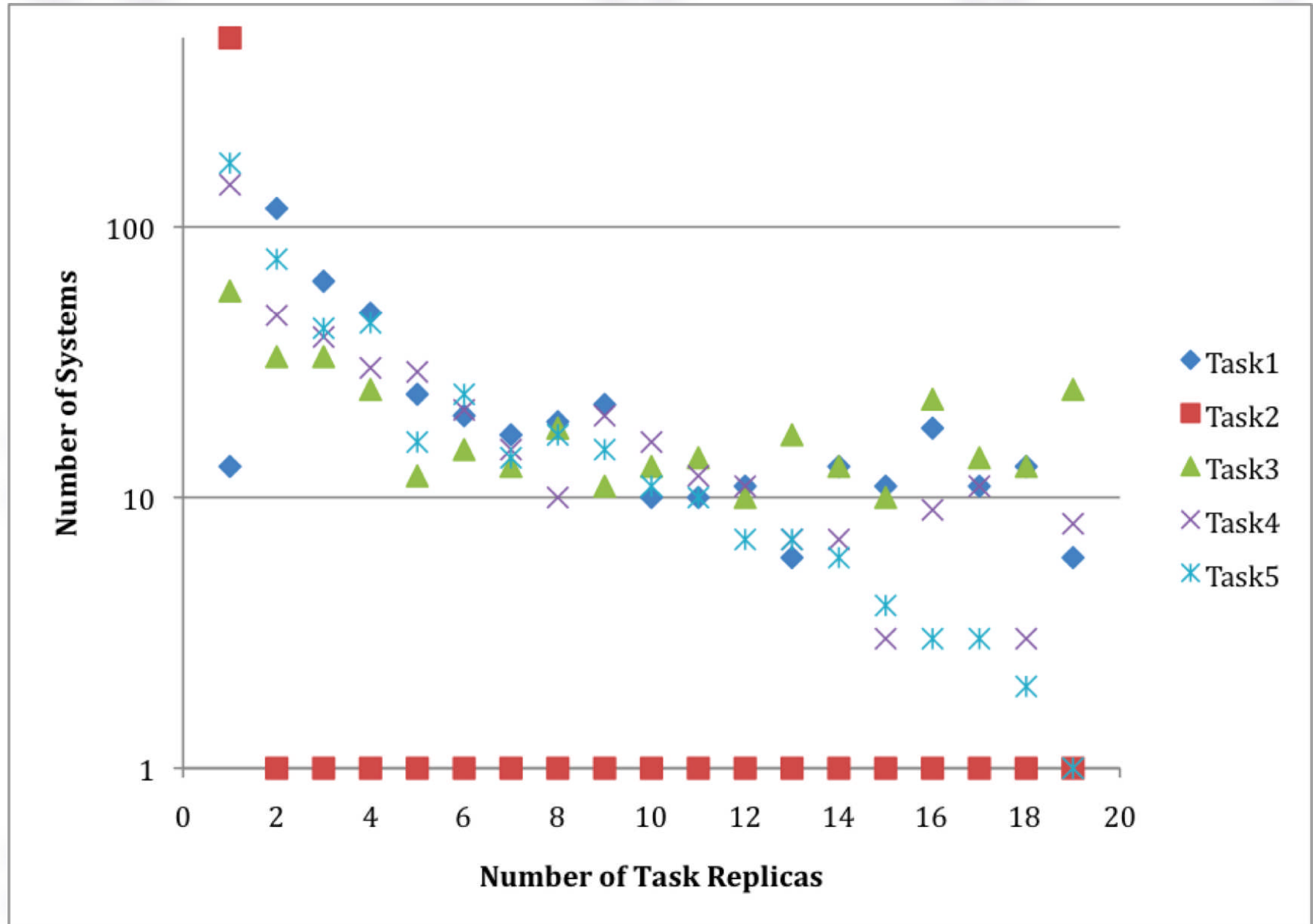
Sample system – three FPGAs

Task	1	2	3	4	5
Criticality	Major	Minor	Hazardous	Catastrophic	Hazardous
Size	256*256	100*266	200*365	150*275	150*50

Results – 500 runs

	Min	Max	Average
No FPGAs	3	8	3.36
Fitness	0.36	0.56	0.390
Task 1	1	46	8.324
Task 2	1	8	1.014
Task 3	1	42	12.48
Task 4	1	47	7.804
Task 5	1	54	5.31

311 acceptable solutions



Sample System 2 – Three FPGAs

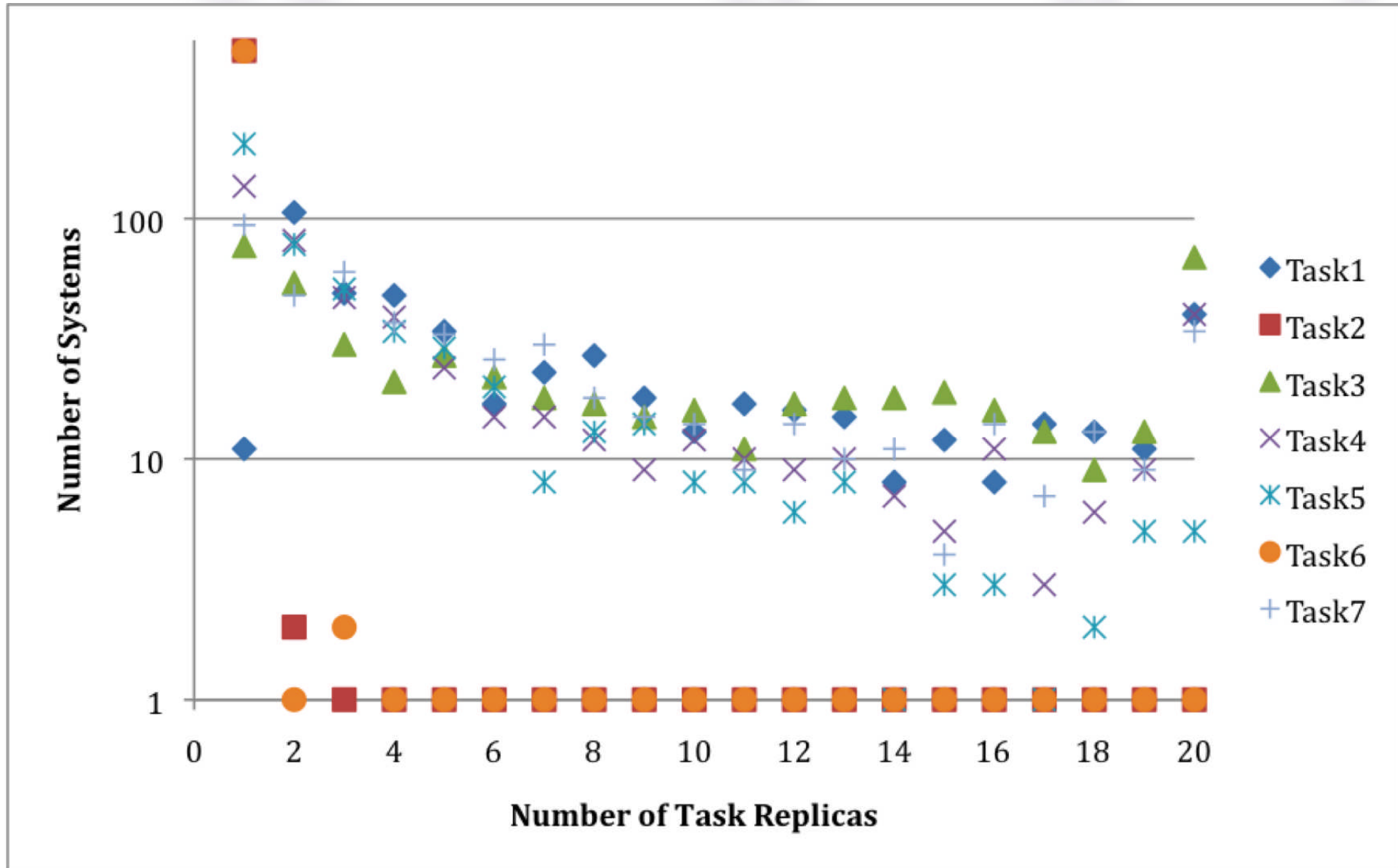
Task	1	2	3	4	5	6	7
Criticality	Major	Minor	Haz	Cat	Haz	Minor	Cat
Size	256*256	100*266	200*365	150*275	150*50	125*150	135*70

Results – 500 runs

	Min	Max	Average
No FPGAs	3	7	3.346
Fitness	0.343	0.629	0.394
Task 1	1	42	8.262
Task 2	1	2	1.004
Task 3	1	40	9.57
Task 4	1	48	6.6
Task 5	1	28	3.818
Task 6	1	13	1.044
Task 7	1	48	7.474

257 acceptable Solutions

Results – Number Tasks



Discussion

- Results mixed
 - Some systems far too many tasks allocated with no improved benefit in terms of reliability
- Still difficult to meet acceptable targets for high criticality systems
 - Problem with NMR/Voter architecture
 - Need to look at other architectures
- Probability targets very closely aligned with penalties
 - Success/fail criteria very finely balanced
 - Adding single extra task copy can tip balance
 - Hard to hone in on optimum number of tasks

Further Work

- Lots!
- Consider more realistic probability model
- Other dependability attributes
- Different architectures
- Dependencies between tasks
- Other methods
 - E.g. Genetic Algorithms

Software Systems Engineering Initiative

www.ssei.org.uk

