Two fully-funded PhD studentships in the Multicore Programming Group at Imperial College London

Closing date for applications: 31 May 2016

I seek applications for two PhD student positions in my Multicore Programming Group at Imperial College London (http://www.multicore.doc.ic.ac.uk), in the area of correct and efficient programming for concurrent and parallel systems. The Department of Computing at Imperial provides a vibrant and stimulating research environment in the heart of London, with leading research groups working on programming languages, verification and testing. The department is consistently recognised by high research ratings. In the 2014 REF assessment, the department was ranked third (1st in the Research Intensity table published by The Times Higher), and was rated as "Excellent" in the previous national assessment of teaching quality.

The posts are fully-funded (covering fees and stipend) for UK and EU students. Unfortunately, non-EU students are not eligible for this funding.

Applicants should have, or expect to soon be awarded, a Masters-level qualification, with distinction, in Computer Science, or a related discipline.

The start date for the posts can be October 2016 or April 2017.

There is flexibility associated with the topics for these positions, but three broad areas that match the interests of the group are as follows:

Automated testing of many-core language implementations. Many-core languages and APIs for graphics and compute, such as Vulkan, OpenGL and OpenCL, are large and complex. Implementing these programming models correctly is challenging, and recent work (led by my group) has used fuzz testing – automatic generation of random programs – to show that production implementations of OpenCL, from a range of vendors, are unreliable. This work has only scratched the surface of the use of fuzz testing for analysing many-core languages and APIs, and a lot more research is required in this area. A particular open problems is that of designing a reusable fuzzing framework. With a lot of engineering effort one can do a good job of building a fuzzer for a particular language or API, but the fuzzer is typically not reusable nor generalizable to other contexts. More desirable would be a declarative framework for specifying API functions and language constructs, together with constraints expressing their acceptable usage, from which a fuzz testing tool could be automatically generated. An alternative to specifying such constraints explicitly is a learning-based approach, whereby an initially naïve fuzzer generates programs that are ranked as acceptable or not by a human (perhaps with the aid of pre-filtering through static or dynamic analysis), and over time uses machine learning to tend towards generating only acceptable programs that can be used for implementation testing. A research contribution in this area would have wide applicability, beyond the domain of many-core programming.

Relevant prior work from my group and collaborators:

- Many-Core Compiler Fuzzing (PLDI'15): http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2015/PLDI_Fuzzing.pdf

- Metamorphic Testing for (Graphics) Compilers (MET'16):
  http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2016/MET.pdf
- Analysing the Program Analyser (V2025):
  http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2016/V2025.pdf

Reasoning about concurrent software. Concurrent software is notoriously hard to write correctly, as well as very difficult to debug when it goes wrong. This is true at the level of distributed software running across multiple nodes, multi-threaded software running across the multiple cores of a single machine, and data parallel software exploiting the resources provided by a GPU accelerator. My group's recent work has shown the potential for static verification to aid in writing reliable GPU code, as well as systematic and controlled-random testing for finding defects in concurrent and distributed programs. However, many open research questions remain in this area. For example, conducting even semi-automatic verification for GPU software that exploits fine-grained concurrency primitives for high performance is not possible with current techniques, and how to scale systematic and controlled-random concurrency testing methods to scale to large applications in order to find deep bugs is wide open. There is broad scope for theoretical work in this area, e.g. properly understanding the weak memory and execution models of GPU architectures and programming models, as well as tool-building projects and empirical studies.

Relevant prior work from my group and collaborators:
- Exposing Errors Related to Weak Memory in GPU Applications (PLDI'16):
  https://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2016/PLDI.pdf
- GPU Concurrency: Weak Behaviours and Programming Assumptions (ASPLOS'15):
  http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2015/ASPLOS.pdf
- Asynchronous Programming, Analysis and Testing with State Machines (PLDI'15):
  http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2015/PLDI_PSharp.pdf
- Concurrency Testing Using Schedule Bounding: an Empirical Study (PPoPP'14):
  http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2014/PPoPP.pdf

Generation of optimised accelerator code from high-level languages. Low-level accelerator programming, in languages like OpenCL and CUDA, is error-prone and requires a high degree of manual optimization effort. An appealing alternative is to design smart compilers to automatically generate high-performance accelerator code from high-level language descriptions. Despite a growing body of work in this area, there is a lack of understanding related to the factors that influence performance portability of generated code across a heterogeneous range of platforms, and a compilation framework capable of generating high-performance code for multi-core CPUs, many-core GPUs from multiple vendors, as well as FPGA devices, is still out of reach. An exciting PhD topic in this area would involve conducting an empirical study to understand the performance trade-offs between these various platforms, encompassing energy consumption as well as execution time, and using the results of this study to devise new programming language constructs, program transformations and compiler optimisations to maximise performance.

Relevant prior work from my group and collaborators:
- PENCIL: a Platform-Neutral Compute Intermediate Language for Accelerator Programming (PACT'15): https://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2015/PACT.pdf
- The Hitchhiker's Guide to Cross-Platform OpenCL Application Development (IWOCL'16):
  http://www.doc.ic.ac.uk/~afd/homepages/papers/pdfs/2016/IWOCL_Hitchhiker.pdf

How to apply:

Before putting together a full application, please contact me (alastair.donaldson@imperial.ac.uk) to confirm eligibility for funding and suitability of qualifications. I'm happy to answer queries you may have regarding the positions.

To formally apply, follow the Department of Computing at Imperial's standard procedure for PhD applications (http://www.imperial.ac.uk/study/pg/apply/how-to-apply), name me as a potential supervisor, and then email me to let me know you have applied.

As part of the application you need to write a research statement, which you may (but need not) base on one of the topics above, perhaps drawing on open problems raised in some of the papers linked to above. Either way, the research statement should emphasise your own novel ideas and track record as much as possible.

Alastair Donaldson