The continuing renaissance in parallel programming languages

Simon McIntosh-Smith
University of Bristol
Microelectronics Research Group
simonm@cs.bris.ac.uk

University of BRISTOL

Didn't parallel computing use to be a niche?

When I were a lad…

University of BRISTOL

But now parallelism is mainstream

Samsung Exynos 5 Octa:

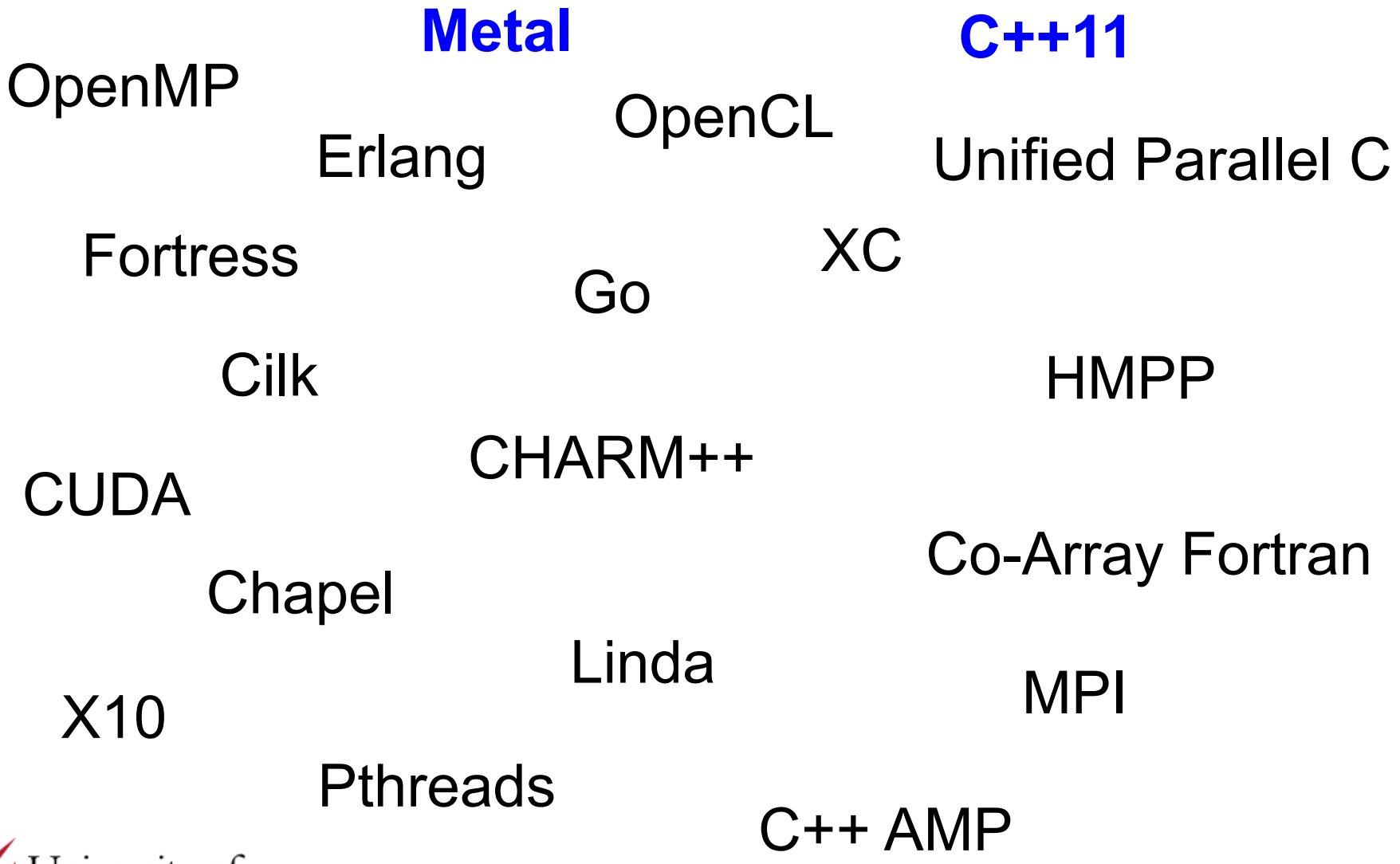
- 4 fast ARM cores and 4 energy efficient ARM cores
- Includes OpenCL programmable GPU from Imagination

HPC scaling to millions of cores

Tianhe-2 at NUDT in China
33.86 PetaFLOPS (33.86x1015), 16,000 nodes
Each node has 2 CPUs and 3 Xeon Phis
3.12 million cores, $390M, 17.6 MW, 720m^2

University of BRISTOL

A renaissance in parallel programming



University of BRISTOL

Groupings of || languages

Partitioned Global Address Space (PGAS):

- Fortress
- X10
- Chapel
- Co-array Fortran
- Unified Parallel C

CSP: XC

Message passing: MPI

Shared memory: OpenMP

GPU languages:

- OpenCL
- CUDA
- HMPP
- **Metal**

Object oriented:

- C++ AMP
- CHARM++

Multi-threaded:

- Cilk
- Go
- **C++11**

Emerging GPGPU standards

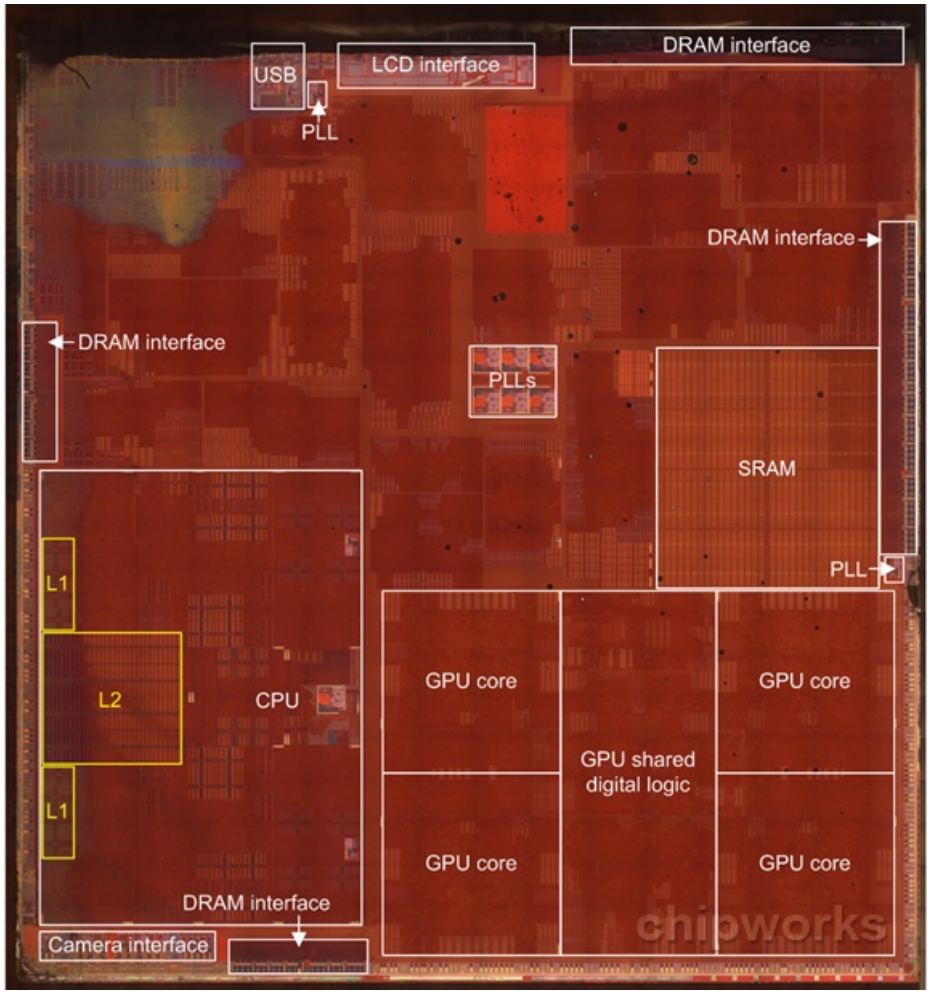
- **OpenCL**, DirectCompute, C++ AMP, …

- Also OpenMP 4.0, OpenACC, CUDA…

Apple's Metal

- A "ground up" parallel programming language for GPUs
- Designed for compute and graphics
  - Potential to replace OpenGL compute shaders, OpenCL/GL interop etc.
- Close to the "metal"
- Low overheads
- "Shading" language based on C++11
- Precompiled shaders

Apple's SoCs highly parallel



Apple A7, courtesy Chipworks

University of BRISTOL

More on Metal

- Currently proprietary (but might be opened?)
- "10X more draw calls per frame"
  - Potentially much better graphical applications
- Focused on iOS (for now?)
- Thin API between the app and hardware
- Targeting latest, newest GPU features
- Reduces frequency of expensive CPU ops
- Predictable performance
- Explicit command submission

Metal

- Can interleave commands for "render", "compute" and "blit" into a single command buffer
- This removes the need for expensive state save/restore between different commands
- Can generate commands in parallel using multiple threads – no atomic locks for improved scalability
- Command encoders generate commands immediately – no deferred state validation

Metal

- Designed for unified memory systems
- Avoids implicit memory copies
- Automatic CPU/GPU coherency model
  - CPU and GPU observe writes at command buffer execution boundaries
  - No explicit CPU cache management required
- Puts more of the synchronisation onus on the programmer, to achieve better performance

Metal's impact

Next Generation OpenGL Initiative

- **Ground up design of open standard for high-efficiency access to graphics and compute on modern GPUs!**
  - Fast-paced work on detailed proposals and designs are already underway
- **Explicit application control over GPU and CPU workloads**
  - High performance and predictability
- **Multithreading-friendly API**
  - Greatly reduced overhead
- **Common shading language intermediate representation**
  - Simpler than a source language to improve shader reliability and portability
  - Good target for machine-generated shaders and high-level languages
  - Some IP protection for shader authors as don't have to ship shader source
  - Can use common compiler front end across multiple platforms

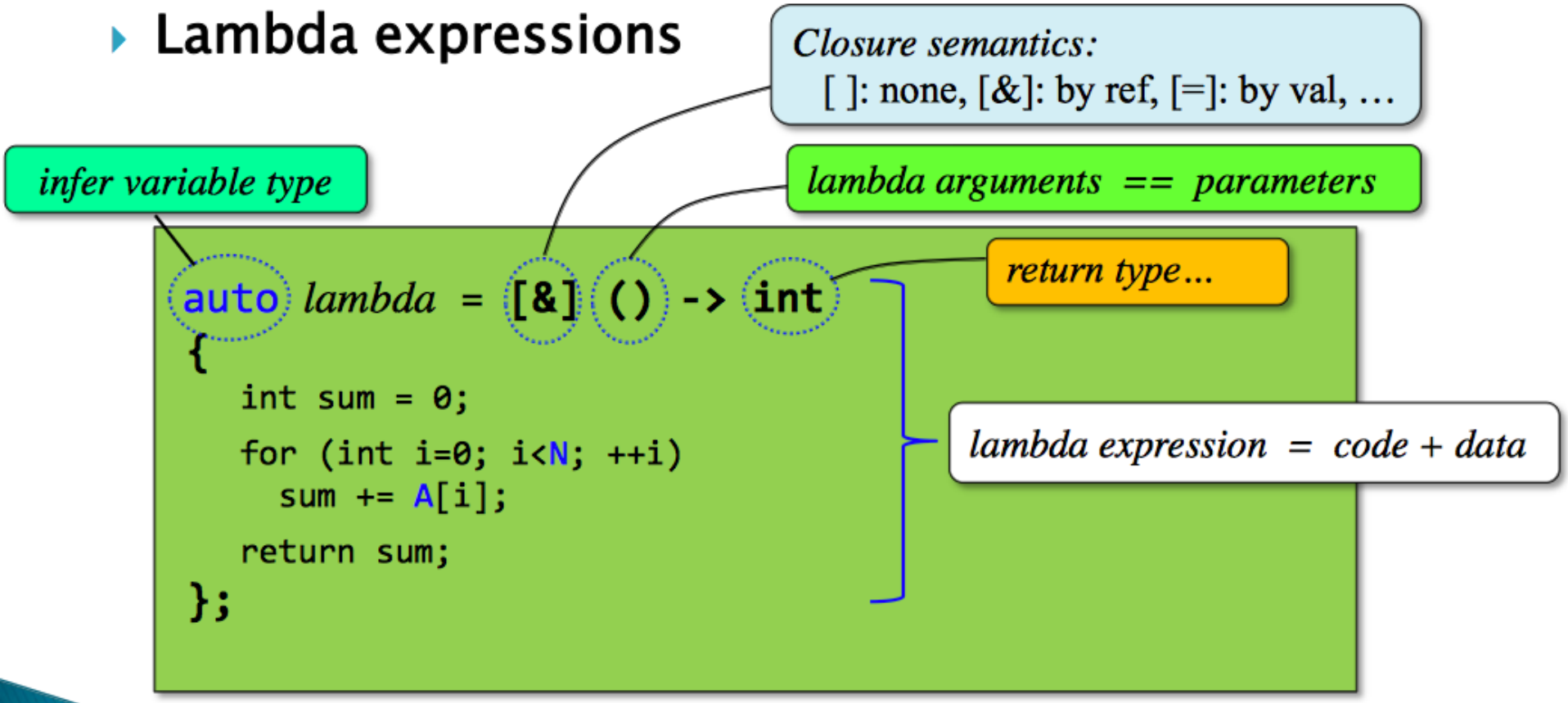
KHRONOS GROUP

https://www.khronos.org/news/press/khronos-group-announces-key-advances-in-opengl-ecosystem

C++11 new parallelism features

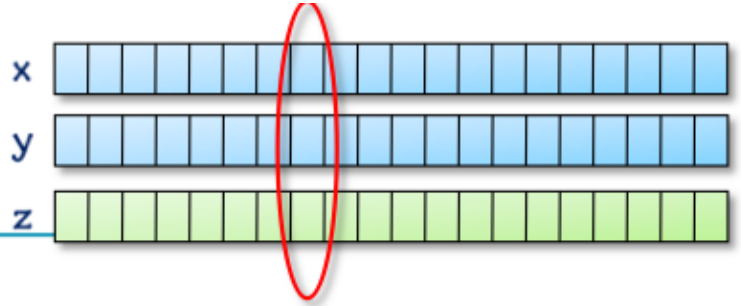
- std::thread class now part of standard C++ library
- Adds lambda expressions (anonymous functions)
- Lots of other activity exploring Parallelism and Concurrency support for C++14 and beyond

- Type inference
- Lambda expressions



From: http://sc13.supercomputing.org/sites/default/files/prog105/prog105.pdf

Example: saxpy



- Saxpy == ***Scalar Alpha X Plus Y***
  - *Scalar multiplication and vector addition*

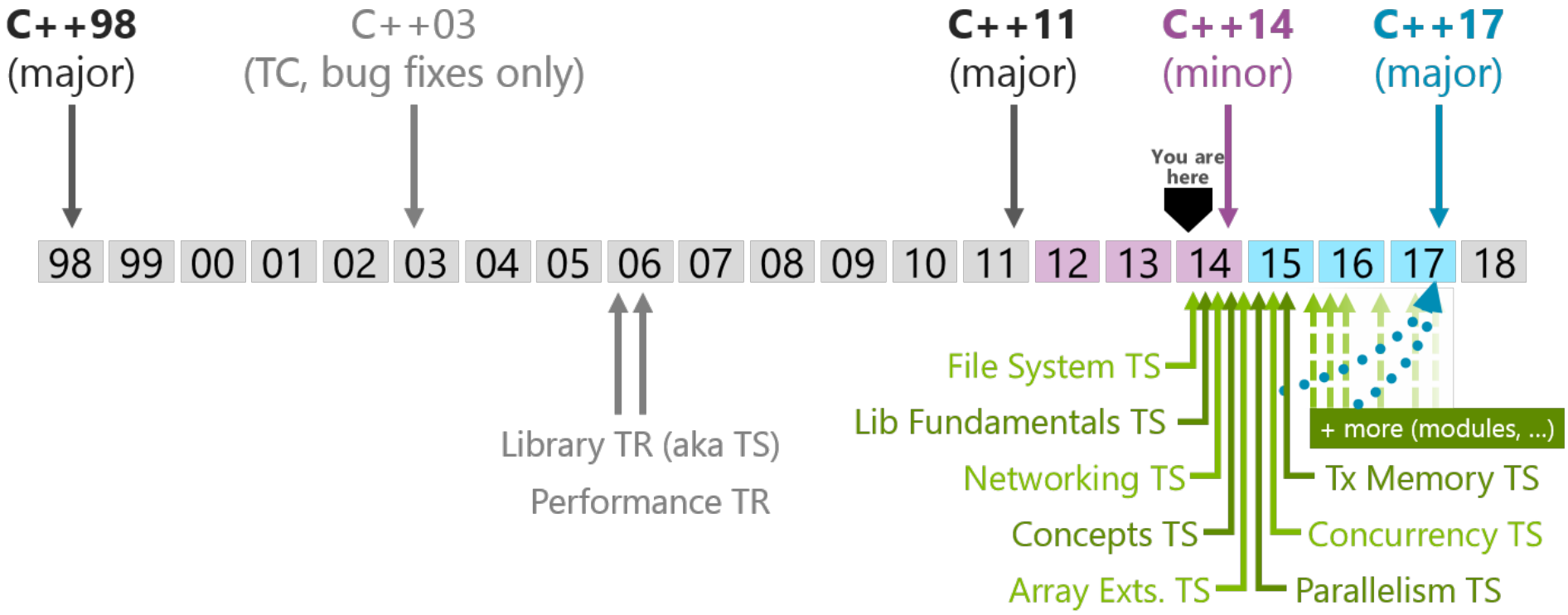
```
for (int i=0; i<n; i++)
   z[i] = a * x[i] + y[i];
```

Parallel

```
auto code = [&](int start, int end) -> void
{
  for (int i = start; i < end; i++)
    z[i] = a * x[i] + y[i];
};


thread t1(code, 0   /*start*/, N/2 /*end*/);
thread t2(code, N/2 /*start*/, N   /*end*/);
```

Where next for C++?



From: https://isocpp.org/std/status

Summary

- Parallel languages are going through a renaissance
- Not just for the niche high end any more
- No silver bullets, lots of “wheel reinventing”
- In HPC, many-core processors are being adopted quickly at the high-end; in embedded systems, heterogeneous is "the new normal"
- Standards like OpenCL and OpenGL are competing with vendor proprietary APIs and with the march of C++1X

http://www.cs.bris.ac.uk/Research/Micro/