The resurgence of parallel programming languages

Jamie Hanlon &

Simon McIntosh-Smith

University of Bristol

Microelectronics Research Group

hanlon@cs.bris.ac.uk

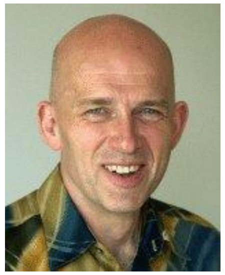
The Microelectronics Research Group at the University of Bristol

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

University of BRISTOL

The team

Simon McIntosh-Smith
Head of Group


Prof David May

Prof Dhiraj Pradhan

Dr Jose Nunez-Yanez

Dr Kerstin Eder

Dr Simon Hollis

Dr Dinesh Pamunuwa

7 tenured staff, 6 research assistants, 16 PhD students

Group expertise

Energy Aware COmputing (EACO):

- Multi-core and many-core computer architectures
  - Inmos, XMOS, ClearSpeed, Pixelfusion, …
- Algorithms for heterogeneous architectures (GPUs, OpenCL)
- Electronic and Optical Network on Chip (NoC)
- Reconfigurable architectures (FPGA)
- Design verification (formal and simulation-based), formal specification and analysis
- Silicon process variation
- Fault tolerant design (hardware and software)
- Design methodologies, modelling & simulation of MNT based structures and systems

Overview

- Parallelism in computing
- Overview and discussion of two current parallel languages
  - Chapel (HPC)
  - OpenCL (desktop/embedded)
- Moving forward
  - Heterogeneous System Architecture
  - Research into scalable general purpose parallel architectures

Didn’t parallel computing use to be a niche?

A long history in HPC…

But now parallelism is mainstream

Quad-core ARM Cortex A9 CPU

Quad-core SGX543MP4+ Imagination GPU

University of BRISTOL

HPC stronger than ever

- 705,024 SPARC64 processor cores delivering 10.51 petaflops (10 quadrillion calculations per second)
- No GPUs or accelerators
- 9.9 MW

Increasing use of GPUs at high end

- Tianhe-1A in Tianjin, China (2nd in Top500)
- 2.6 petaflops
- 14,336 Intel 2.93 GHz CPUs (57,334 cores)
- **7,168 NVIDIA Tesla M2050 GPUs (100,000 cores)**
- 4 MW power consumption

Big computing is mainstream too

CANADA
WASHINGTON
IDAHO
Seattle
Wenatchee
Quincy
90
Columbia R.
Pacific Ocean
82
Portland
The Dalles
84
Salem
Snake R.
OREGON
5
Miles
0
100

http://www.nytimes.com/2006/06/14/technology/14search.html

Report: Google Uses About 900,000 Servers (Aug 1st 2011)

http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers/

A renaissance in parallel programming

CSP
- Erlang
- Occam-pi
- XC

GPGPU
- OpenCL
- CUDA
- HMPP
- OpenACC

Message-passing
- MPI

Multi-threaded
- OpenMP
- Cilk
- Go

Object-orientated
- C++ AMP
- CHARM++

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

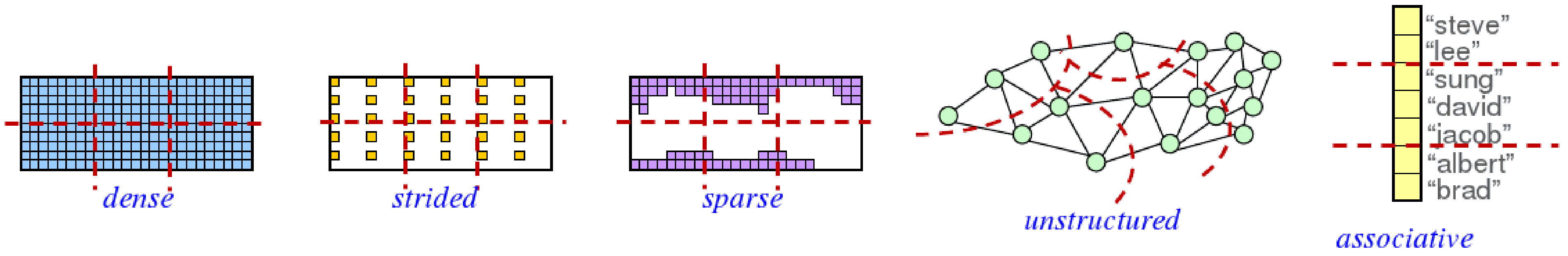
Chapel

Chapel

- Cray development funded by DARPA as part of HPCS program
- Partitioned global address space (PGAS) language
  - Central abstraction is a global array partitioned across a system
- Programmer control of locality by allowing explicit affinity of both tasks and data to *locales*

Arrays and distribution

- An array is a general concept and can be declared with different domains
- Can be distributed with a domain map
  - Standard maps and can be user-defined
- Computation can remain the same regardless of a specific distribution



Chapel's data parallelism

- Zippered forall:

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c
```

  - loop body sees ith element from each iteration
- Works over:
  - distributed arrays
  - arrays with different distributions
  - user-defined iterators – A,B,C could be trees or graphs

MPI+OpenMP

```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

```
  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory
        (%d).\n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 0.0;
  }

  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

Composition in Chapel

- Data parallelism

```
cobegin {
  forall (a, b, c) in (A, B, C) do
    a = b + alpha * c;
  forall (d, e, f) in (D, E, F) do
    d = e + beta * f;
}
```

- Task parallelism nested in data parallelism

```
forall a in A {
  if a == 0 then
    begin a = f(a)
  else
    a = g(a)
}
```

Issues with Chapel

- HPC-orientated: not suitable for general programming, e.g. embedded platforms
- Locales support only a single level hierarchy
- No load balancing/dynamic resource management
- Too high level? Is it a good abstraction of a parallel machine?

OpenCL

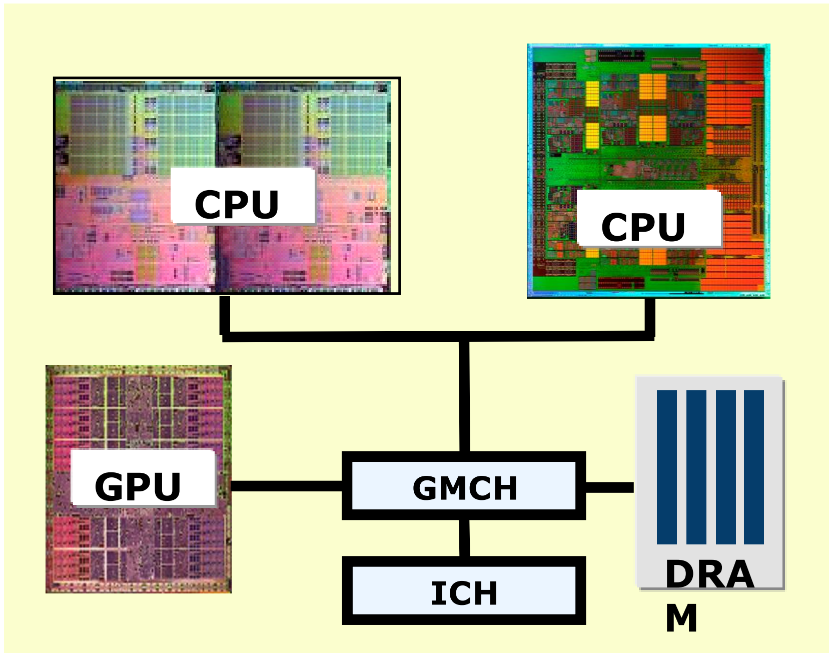
(Open Computing Language)

OpenCL

- Open standard for portable, parallel programming of heterogeneous systems
- Lets programmers write a single **portable** program that uses **all** resources in the heterogeneous platform

A modern system includes:

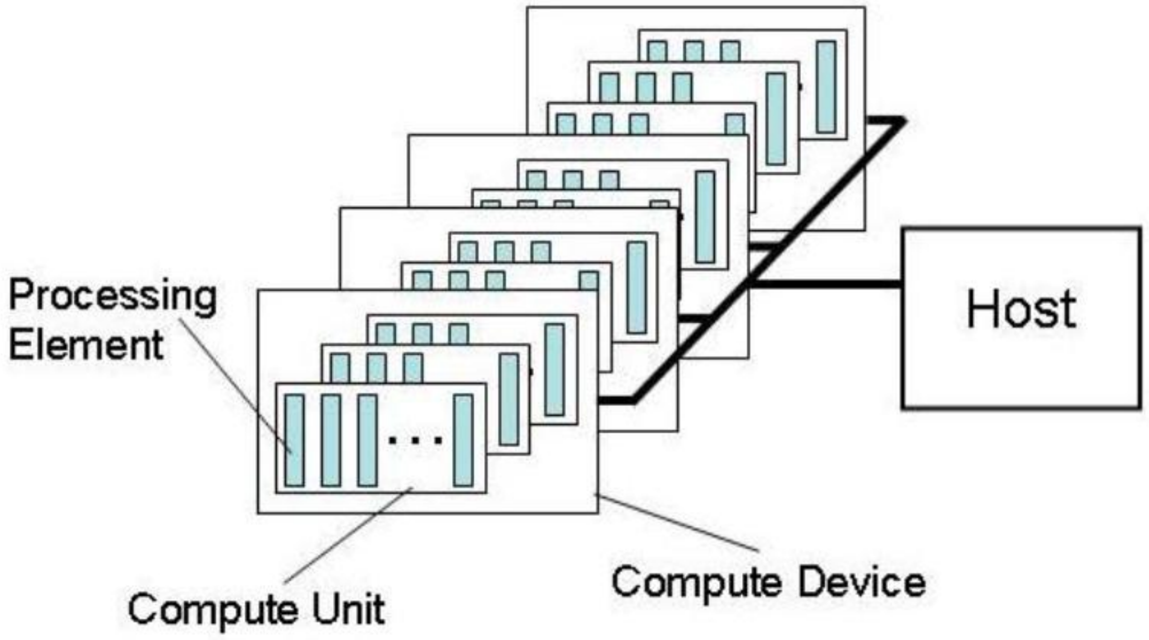
- One or more CPUs
- One or more GPUs
- DSP processors
- …other devices?



GMCH = graphics memory control hub

ICH = Input/output control hub

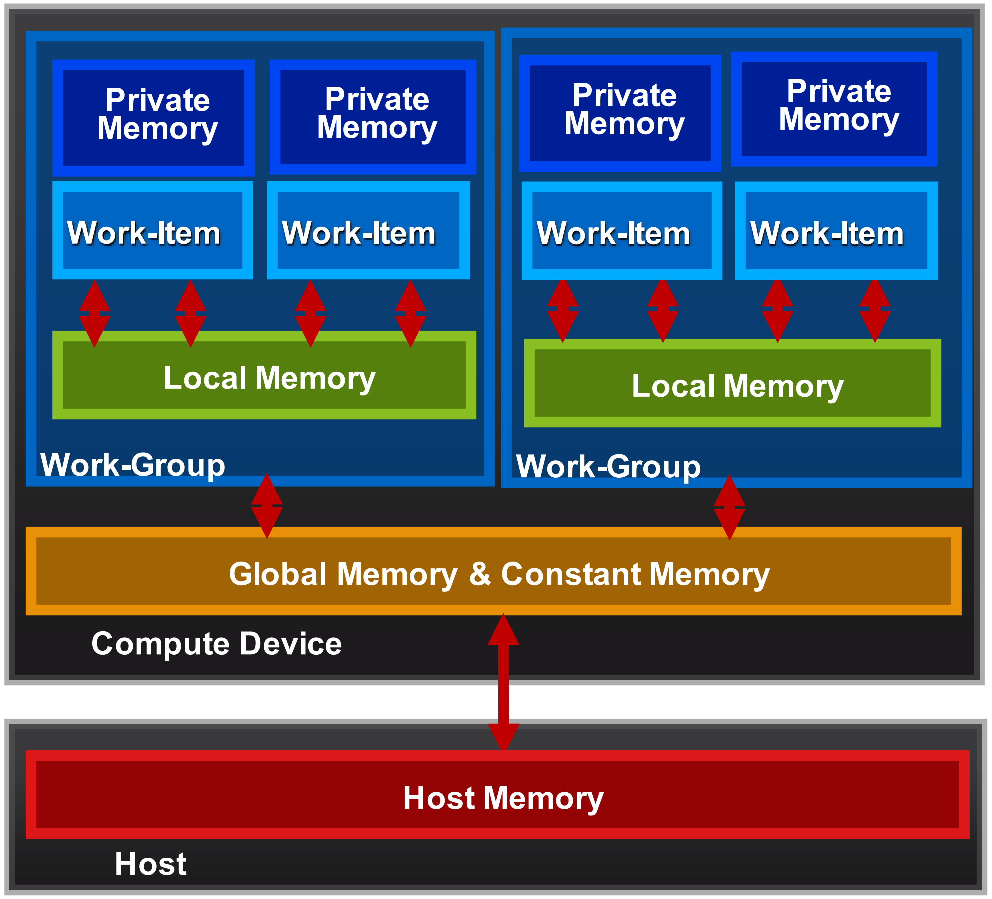
OpenCL platform model



- One Host + one or more Compute Devices
  - Each Compute Device is composed of one or more Compute Units
  - Each Compute Unit is further divided into one or more Processing Elements

OpenCL memory model

- Private Memory
  - Per Work-Item
- Local Memory
  - Shared within a Work-Group
- Global / Constant Memories
  - Visible to all Work-Groups
- Host Memory
  - On the CPU



Memory management is explicit
You must move data from host → global → local *and* back

The BIG idea behind OpenCL

- Replace loops with functions (a kernel) executing at each point in a problem domain (index space).
- E.g., process a 1024 x 1024 image with one kernel invocation per pixel or 1024 x 1024 = 1,048,576 kernel executions

Traditional loops

```
void
trad_mul(const int n,
         const float *a,
         const float *b,
         float *c) {
  int i;
  for (i=0; i<n; i++)
    c[i] = a[i] * b[i];
}
```

Data parallel OpenCL

```
kernel void
dp_mul(global const float *a,
       global const float *b,
       global float *c) {
  int id = get_global_id(0);

  c[id] = a[id] * b[id];

} // execute over "n" work-items
```

Host program boilerplate

```
// create the OpenCL context on a GPU device
cl_context = clCreateContextFromType(0,
   CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);

// get the list of GPU devices associated with context
clGetContextInfo(context, CL_CONTEXT_DEVICES, 0,
                                    NULL, &cb);
devices = malloc(cb);
clGetContextInfo(context, CL_CONTEXT_DEVICES, cb,
   devices, NULL);

// create a command-queue
cmd_queue = clCreateCommandQueue(context, devices[0],
   0, NULL);

// allocate the buffer memory objects
memobjs[0] = clCreateBuffer(context, CL_MEM_READ_ONLY |
   CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*n, srcA,
   NULL);}
memobjs[1] = clCreateBuffer(context,CL_MEM_READ_ONLY |
   CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*n, srcB,
   NULL);
memobjs[2] = clCreateBuffer(context,CL_MEM_WRITE_ONLY,
                         sizeof(cl_float)*n, NULL,
   NULL);

// create the program
program = clCreateProgramWithSource(context, 1,
   &program_source, NULL, NULL);
```

```
// build the program
err = clBuildProgram(program, 0, NULL, NULL, NULL,
   NULL);

// create the kernel
kernel = clCreateKernel(program, “vec_add”, NULL);

// set the args values
err  = clSetKernelArg(kernel, 0, (void *) &memobjs[0],
                                 sizeof(cl_mem));
err |= clSetKernelArg(kernel, 1, (void *)&memobjs[1],
                                 sizeof(cl_mem));
err |= clSetKernelArg(kernel, 2, (void *)&memobjs[2],
                                  sizeof(cl_mem));
// set work-item dimensions
global_work_size[0] = n;

// execute kernel
err = clEnqueueNDRangeKernel(cmd_queue, kernel, 1,
   NULL, global_work_size, NULL, 0, NULL, NULL);

// read output array
err = clEnqueueReadBuffer(context, memobjs[2], CL_TRUE,
   0, n*sizeof(cl_float), dst, 0, NULL, NULL);
```

Host program boilerplate

```
// create the OpenCL context on a GPU device
cl_context = clCreateContextFromType(0,
   CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);

// get the list of GPU devices associated with context
clGetContextInfo(context, CL_CONTEXT_DEVICES, 0,
```

Define platform and queues

```
c
clGetContextInfo(context, CL_CONTEXT_DEVICES, cb,
   devices, NULL);

// create a command-queue
cmd_queue = clCreateCommandQueue(context, devices[0],
   0, NULL);
```

```
// allocate the buffer memory objects
memobjs[0] = clCreateBuffer(context, CL_MEM_READ_ONLY |
   CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*n, srcA,
   NULL);
memo                                          ONLY |
   C                                          cB,
   NULL);
memobjs[2] = clCreateBuffer(context,CL_MEM_WRITE_ONLY,
                            sizeof(cl_float)*n, NULL,
   NULL);
```

Define Memory objects

```
// cre
progra                                        1,
   &pr
```

Create the program

```
// build
err = clB                                    NULL,
   NULL);
```

Build the program

```
// create the kernel
kernel = clCreateKernel(program, "vec_add", NULL);

//
err                                        js[0],

err |= clSetKernelArg(kernel, 1, (void *)&memobjs[1],
                                sizeof(cl_mem));
err |= clSetKernelArg(kernel, 2, (void *)&memobjs[2],
                                 sizeof(cl_mem));
```

Create and setup kernel

```
// set work-item dimensions
global_work_size[0] = n;

// execu
err = clEnqueueNDRangeKernel(cmd_queue, kernel, 1,
   NULL, global_work_size, NULL, 0, NULL, NULL);
```

Execute the kernel

```
/
e                                              E,
```

Read results back to the host

Issues with OpenCL

- Low level
- It does not compose
  - Disjoint memory address spaces (local/global)
  - Barriers
- It provides no resource management
  - Kernels are a statically allocated resource

Heterogeneous System Architecture (HSA)

HSA overview

- Announced recently by AMD as new open architecture specification
  - HSAIL virtual ISA
  - HSA memory model
  - HSA dispatch
- Designed to support CPU-GPU integration in APU
- Provides an optimised platform architecture for OpenCL
- Already being adopted by other vendors starting with ARM

HSA features

- Integration of CPU and GPU in silicon
- Unified address space for CPU and GPU
- Potentially even GPU context switching!
- HSA programming model introduces PGAS-style distributed arrays
  - Memory hierarchy abstraction to address function composition
- First class barrier objects

HSA Intermediate Layer (HSAIL)

- Virtual ISA for parallel programs
- Similar idea to LLVM IR - a good target for compilers
- Finalised to specific ISA by a JIT compiler
- Features:
    - Explicitly parallel
    - Support for exceptions, virtual functions and other high-level features
    - Syscall methods (I/O, printf etc.)
    - Debugging support

HSA memory model

- Compatible with C++11, Java and .NET memory models
- Relaxed consistency

HSA dispatch

- HSA designed to enable heterogeneous task queuing
  - A work queue per core
  - Distribution of work into queues
  - Load balancing by work stealing

Problems in the long term

- Programming model split between two architectures
- Cost of data movement between CPU and GPU will be reduced but still present
- Not scalable beyond a single chip

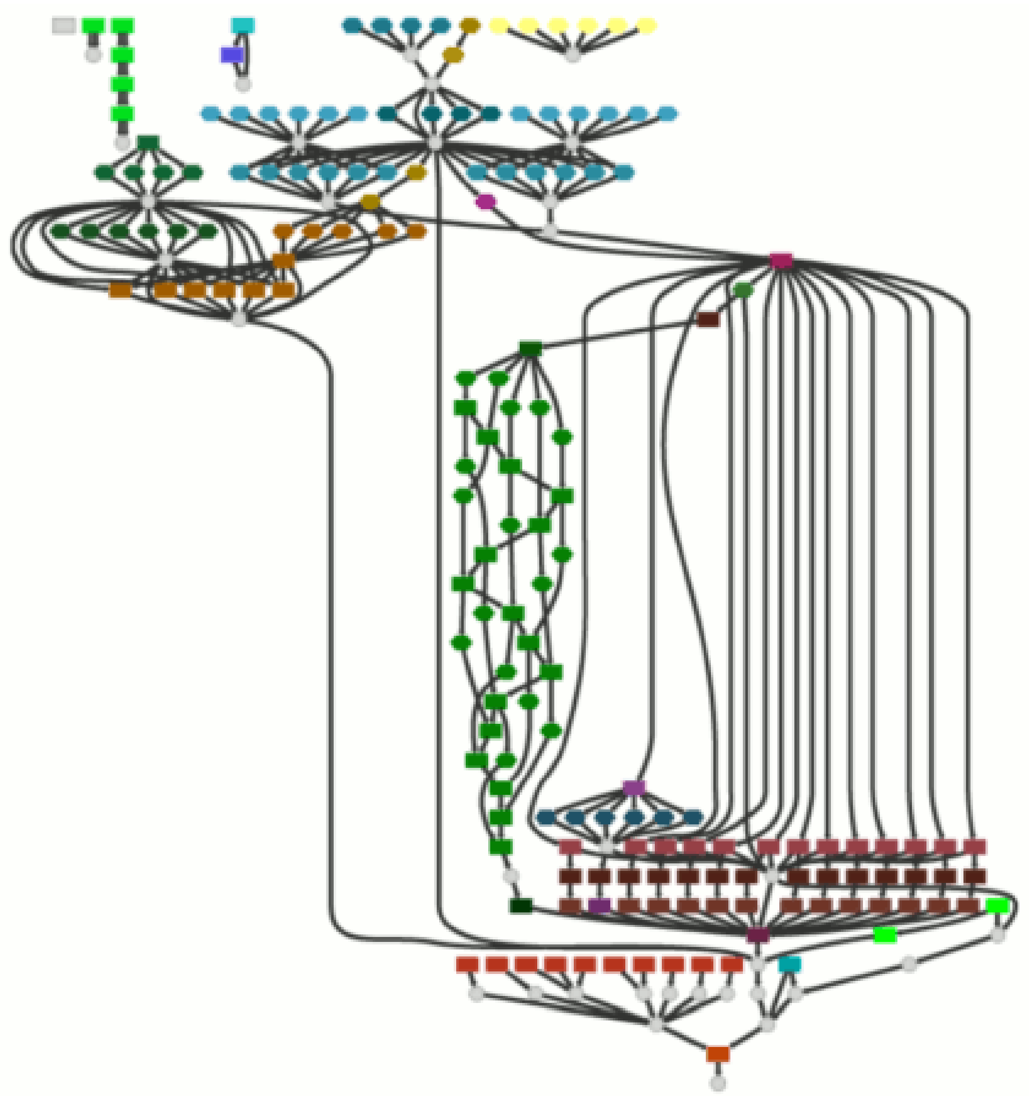
Research into scalable general purpose architectures

Simplify programming

Need general purpose parallel processors to simplify programming

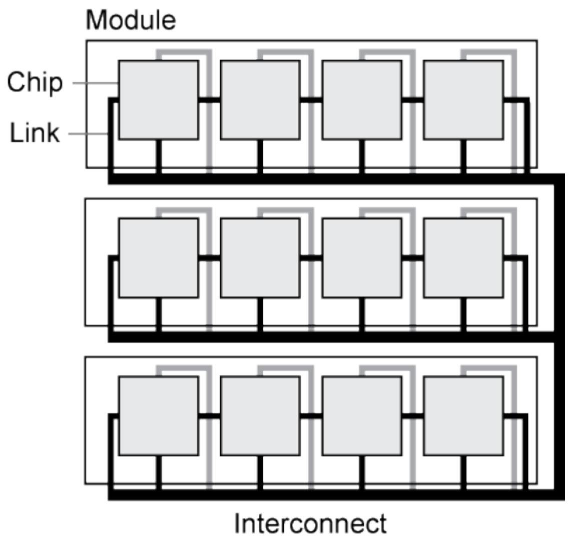
Must support many algorithms, even within a single application
e.g. Task (farms, pipeline) and data parallelism

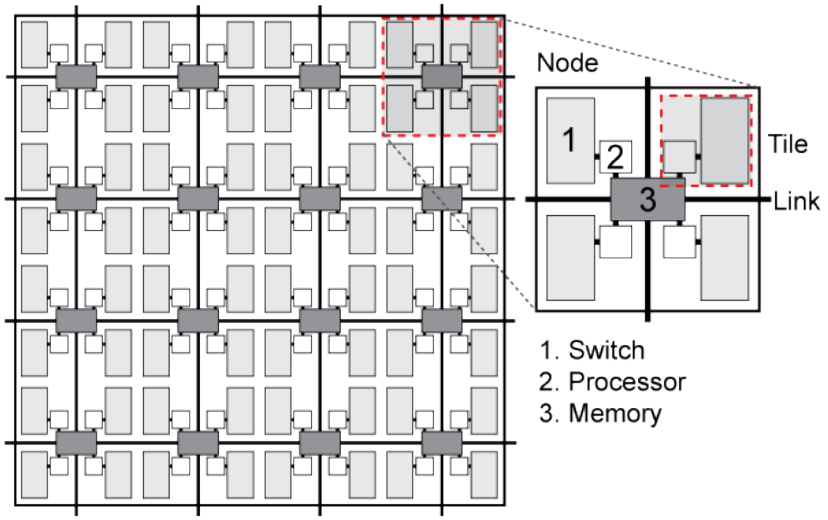
Performance must be comparable to special-purpose devices



Performance must be scalable

- 1 - 1000 of cores per chip
  - Potentially millions of cores in a system
- Regular tiled implementation on chips, modules and boards





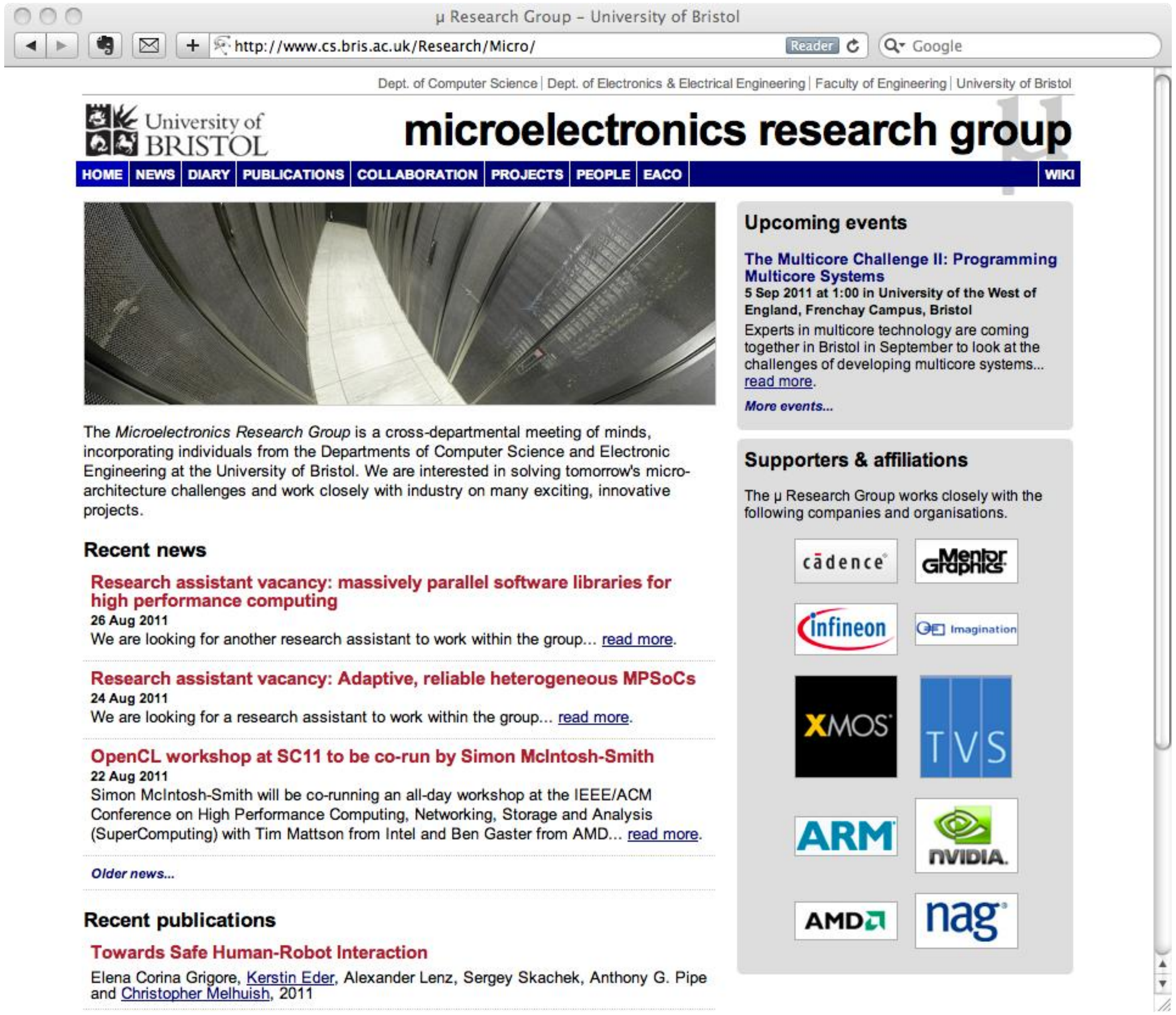
Interconnect

- Must provide low latency, high throughput communication
- This must scale well with the number of processors
- Clos & hypercube networks provide these properties but it is assumed they are prohibitively difficult to build
    - Low dimensional meshes seem to be the convention
- Potential in new technology: 3D stacking, silicon substrates, optical interconnections, …

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, GPUs being adopted quickly at the high-end
- In embedded computing, OpenCL gaining ground
- Movement towards high level general purpose models of parallelism

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


μ Research Group – University of Bristol
http://www.cs.bris.ac.uk/Research/Micro/
Reader
Google
Dept. of Computer Science | Dept. of Electronics & Electrical Engineering | Faculty of Engineering | University of Bristol
University of BRISTOL
HOME NEWS DIARY PUBLICATIONS COLLABORATION PROJECTS PEOPLE EACO WIKI

microelectronics research group

The *Microelectronics Research Group* is a cross-departmental meeting of minds, incorporating individuals from the Departments of Computer Science and Electronic Engineering at the University of Bristol. We are interested in solving tomorrow's micro-architecture challenges and work closely with industry on many exciting, innovative projects.

Recent news

Research assistant vacancy: massively parallel software libraries for high performance computing
26 Aug 2011
We are looking for another research assistant to work within the group... read more.

Research assistant vacancy: Adaptive, reliable heterogeneous MPSoCs
24 Aug 2011
We are looking for a research assistant to work within the group... read more.

OpenCL workshop at SC11 to be co-run by Simon McIntosh-Smith
22 Aug 2011
Simon McIntosh-Smith will be co-running an all-day workshop at the IEEE/ACM Conference on High Performance Computing, Networking, Storage and Analysis (SuperComputing) with Tim Mattson from Intel and Ben Gaster from AMD... read more.

Older news...

Recent publications

Towards Safe Human-Robot Interaction
Elena Corina Grigore, Kerstin Eder, Alexander Lenz, Sergey Skachek, Anthony G. Pipe and Christopher Melhuish, 2011

Upcoming events

The Multicore Challenge II: Programming Multicore Systems
5 Sep 2011 at 1:00 in University of the West of England, Frenchay Campus, Bristol
Experts in multicore technology are coming together in Bristol in September to look at the challenges of developing multicore systems... read more.

More events...

Supporters & affiliations

The μ Research Group works closely with the following companies and organisations.