OP2 – an open-source library for unstructured grid applications

Mike Giles, Gihan Mudalige

mike.giles@maths.ox.ac.uk

Oxford University Mathematical Institute

Oxford e-Research Centre

Outline

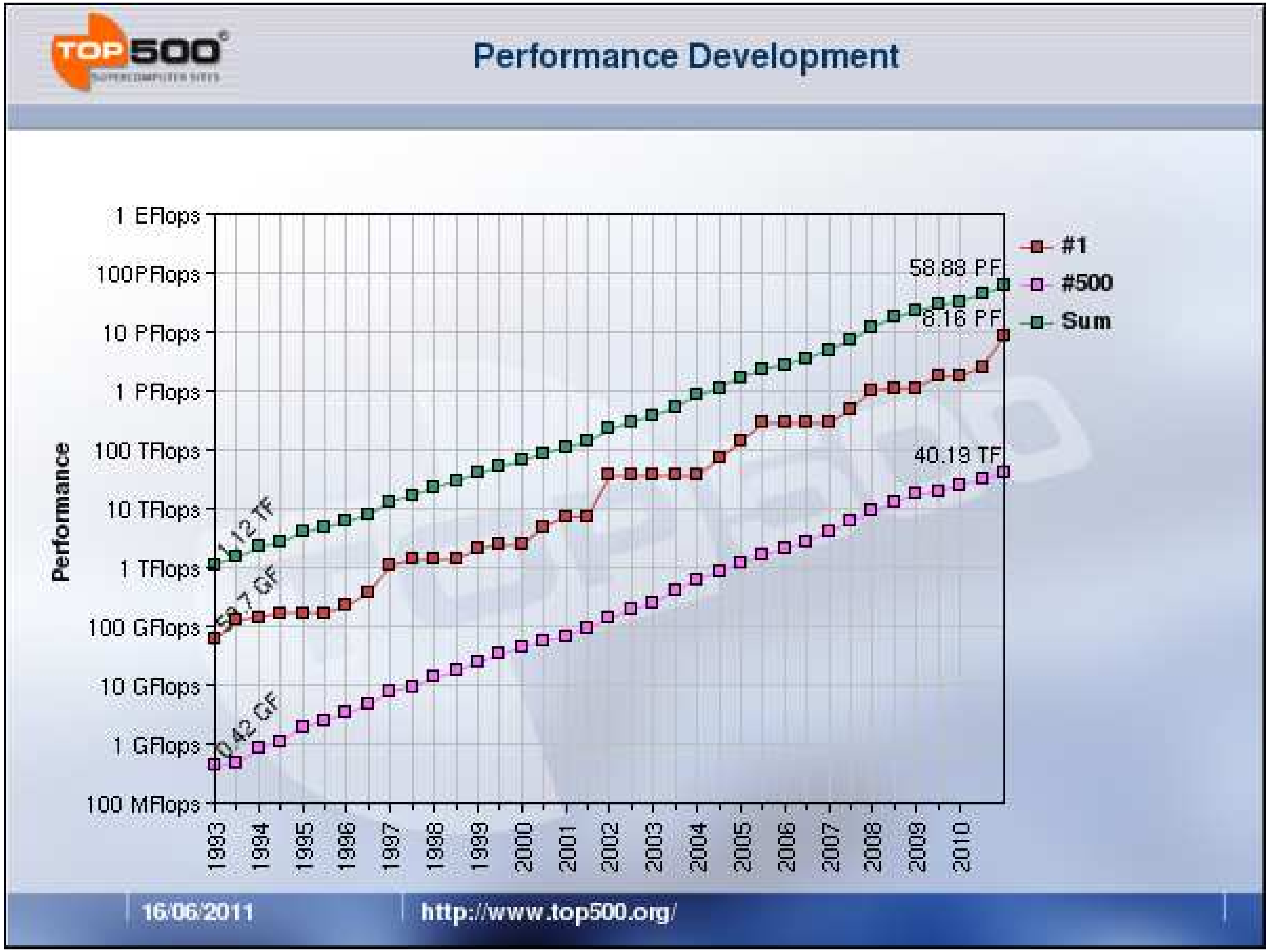
- trends in computing
- opportunity, challenges, context
- user perspective (i.e. application developer)
  - API
  - build process
- some implementation issues
- some performance results

Computing

Computing used to be simple (1 CPU, 1 core, 1 thread) but those days are long gone:

- a server now has 2-4 CPUs, each with 2-12 cores
- with 2 threads often running on each core, this gives a total of up to 96 threads working in parallel on a single application
- unfortunately, the programmer has to take responsibility for most of this – can't just rely on the compiler to take care of it
- the good news – because of this parallelism, the overall compute speed is still doubling every 18 months

Computing

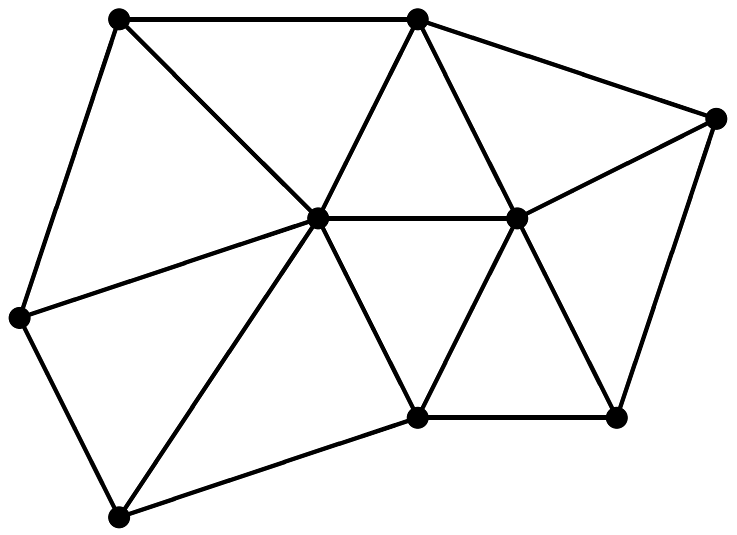

TOP500 SUPERCOMPUTER SITES
Performance Development
Performance
1 EFlops
100PFlops
10 PFlops
1 PFlops
100 TFlops
10 TFlops
1 TFlops
100 GFlops
10 GFlops
1 GFlops
100 MFlops
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
58.88 PF
8.16 PF
40.19 TF
1.12 TF
59.7 GF
0.42 GF
#1
#500
Sum
16/06/2011
http://www.top500.org/

Computing

And if you thought that was complicated . . .

- graphics chips (GPUs) originally designed for graphics and computer games, are now programmable and capable of high performance computing
- NVIDIA GPUs have up to 512 cores, arranged as 16 units each with 32 cores working as a vector unit (i.e. all 32 doing the same operation at the same time but with different data)
- typically lots of threads per core (to hide the effect of delays in fetching data from memory) so often up to 10,000 threads running at the same time on one GPU
- can be quite challenging to do the programming – needs a good understanding of the hardware

Unstructured grids



- a collection of nodes, edges, faces, cells, etc., each addressed by a 1D index
- explicit connectivity – mapping tables define connections from edges to nodes, or faces to cells, etc.
- much harder to parallelise than structured grid applications (not in concept so much as in practice) but a lot of existing literature on the subject

Software Challenge

- Application developers want the benefits of the latest hardware but are very worried about the software development effort, and the expertise required
- Status quo is not really an option – running lots of single-thread MPI processes on multiple CPUs won’t give great performance
- Want to exploit GPUs using CUDA, and CPUs using OpenMP/AVX
- However, hardware is likely to change rapidly in next few years, and developers can not afford to keep changing their software implementation

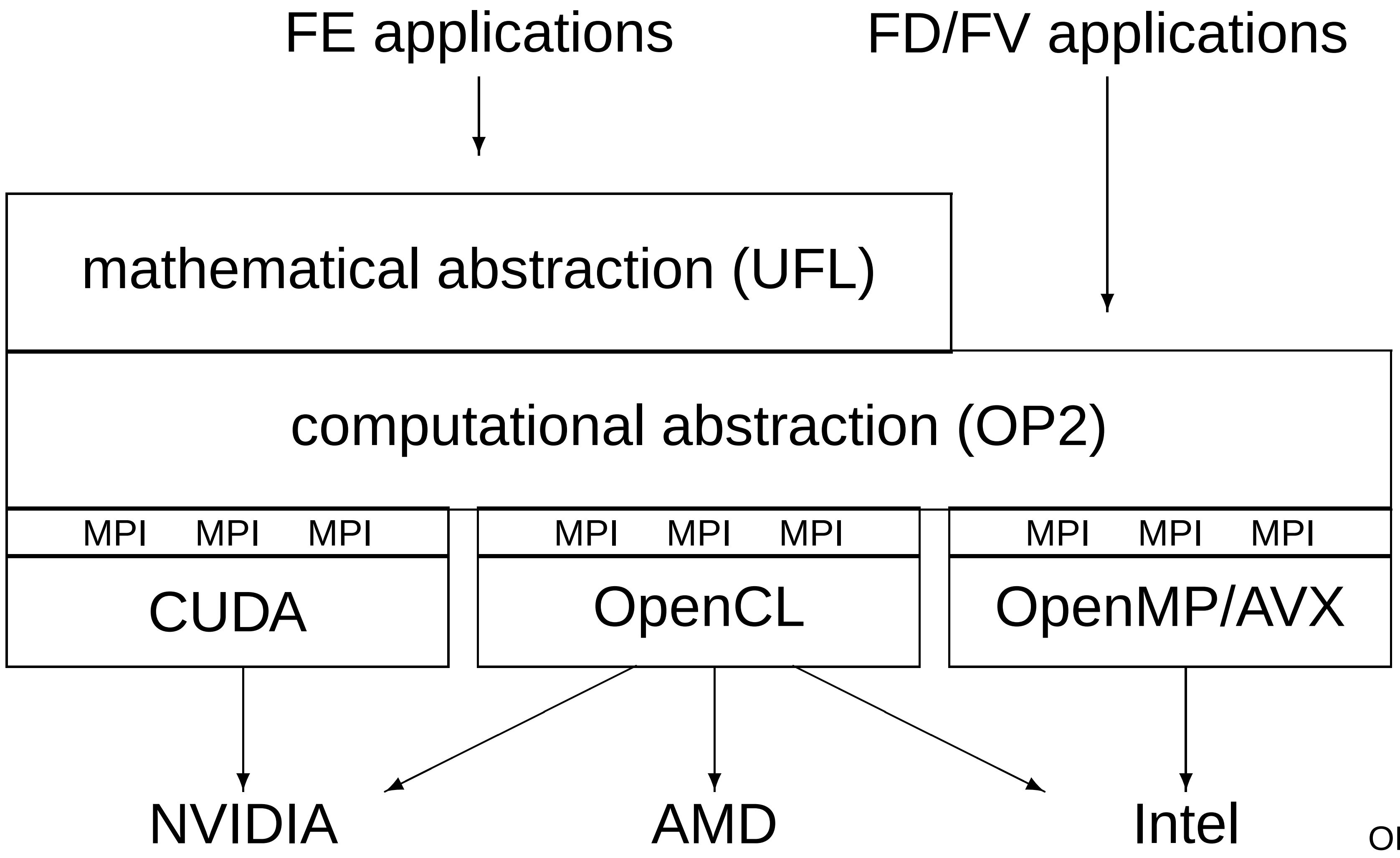
Software Abstraction

To address this challenge, need to move to a suitable level of abstraction:

- separate the user’s specification of the application from the details of the parallel implementation
- aim to achieve application level longevity with the user specification not changing for perhaps 10 years
- aim to achieve near-optimal performance through re-targetting the back-end implementation to different hardware and low-level software platforms

Context

Part of a larger project led by Paul Kelly at Imperial College:
David Ham will talk about the UFL work on Friday



History

OPlus (Oxford Parallel Library for Unstructured Solvers)

- developed for Rolls-Royce 10 years ago
- MPI-based library for HYDRA CFD code on clusters with up to 200 nodes

OP2:

- open source project
- keeps OPlus abstraction, but slightly modifies API
- an “active library” approach with code transformation to generate CUDA for GPUs and OpenMP/AVX for CPUs

OP2 Abstraction

- sets (e.g. nodes, edges, faces)
- datasets (e.g. flow variables)
- mappings (e.g. from edges to nodes)
- parallel loops
  - operate over all members of one set
  - datasets have at most one level of indirection
  - user specifies how data is used
    (e.g. read-only, write-only, increment)

OP2 Restrictions

- set elements can be processed in any order, doesn’t affect result to machine precision
  - explicit time-marching, or multigrid with an explicit smoother is OK
  - Gauss-Seidel or ILU preconditioning is not
- static sets and mappings (no dynamic grid adaptation)

OP2 API

```
void op_init(int argc, char **argv)

op_set op_decl_set(int size, char *name)

op_map op_decl_map(op_set from, op_set to,
                   int dim, int *imap, char *name)

op_dat op_decl_dat(op_set set, int dim,
                   char *type, T *dat, char *name)

void op_decl_const(int dim, char *type,
                   T *dat)

void op_exit()
```

OP2 API

Example of parallel loop syntax for a sparse matrix-vector product:

```
op_par_loop(res,"res", edges,
  op_arg_dat(A,-1,OP_ID, 1,"float",OP_READ),
  op_arg_dat(u, 1,pedge,1,"float",OP_READ),
  op_arg_dat(du,0,pedge,1,"float",OP_INC ));
```

This is equivalent to the C code:

```
for (e=0; e<nedges; e++)
   du[pedge[2*e]] += A[e] * u[pedge[1+2*e]];
```

where each “edge” corresponds to a non-zero element in the matrix A, and pedge gives the corresponding row and column indices.

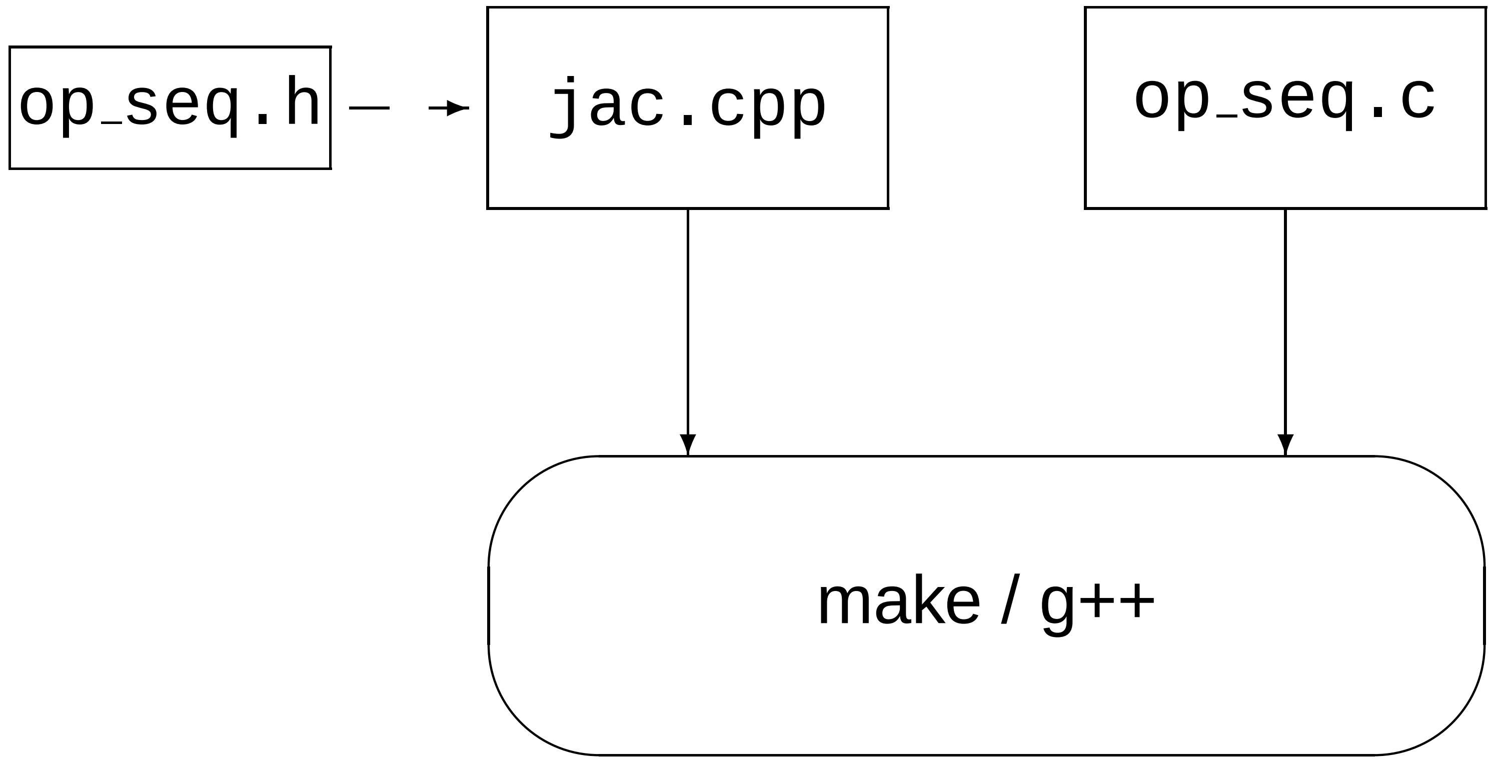
User build processes

Using the same source code, the user can build different executables for different target platforms:

- sequential single-thread CPU execution
  - purely for program development and debugging
  - very poor performance
- CUDA for single GPU
- OpenMP/AVX for multicore CPU systems
- MPI plus any of the above for clusters

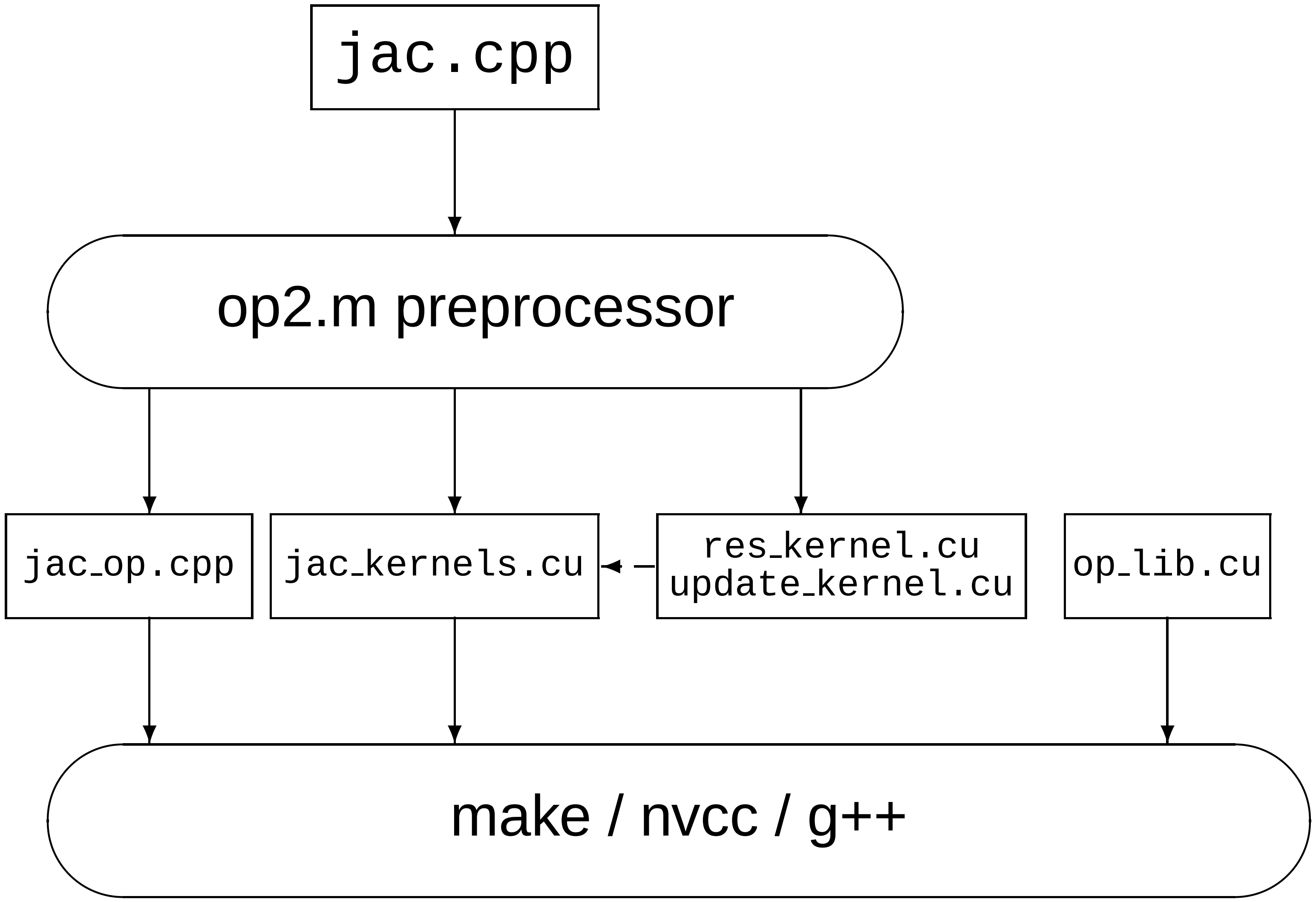
Sequential build process

Traditional build process, linking to a conventional library in which many of the routines do little but error-checking:



CUDA build process

Preprocessor parses user code and generates new code:



GPU Parallelisation

Could have up to 10^6 threads in 3 levels of parallelism:

- MPI distributed-memory parallelism (1-100)
  - one MPI process for each GPU
  - all sets partitioned across MPI processes, so each MPI process only holds its data (and halo)
  - each partition sized to fit within global memory of GPU (up to 6GB)
  - only halos need to be transferred from one GPU to another, via the CPUs
  - hopefully, this will give a balanced implementation – slight possibility that MPI networking will end up being the primary bottleneck, so will work hard to overlap computation and MPI communication

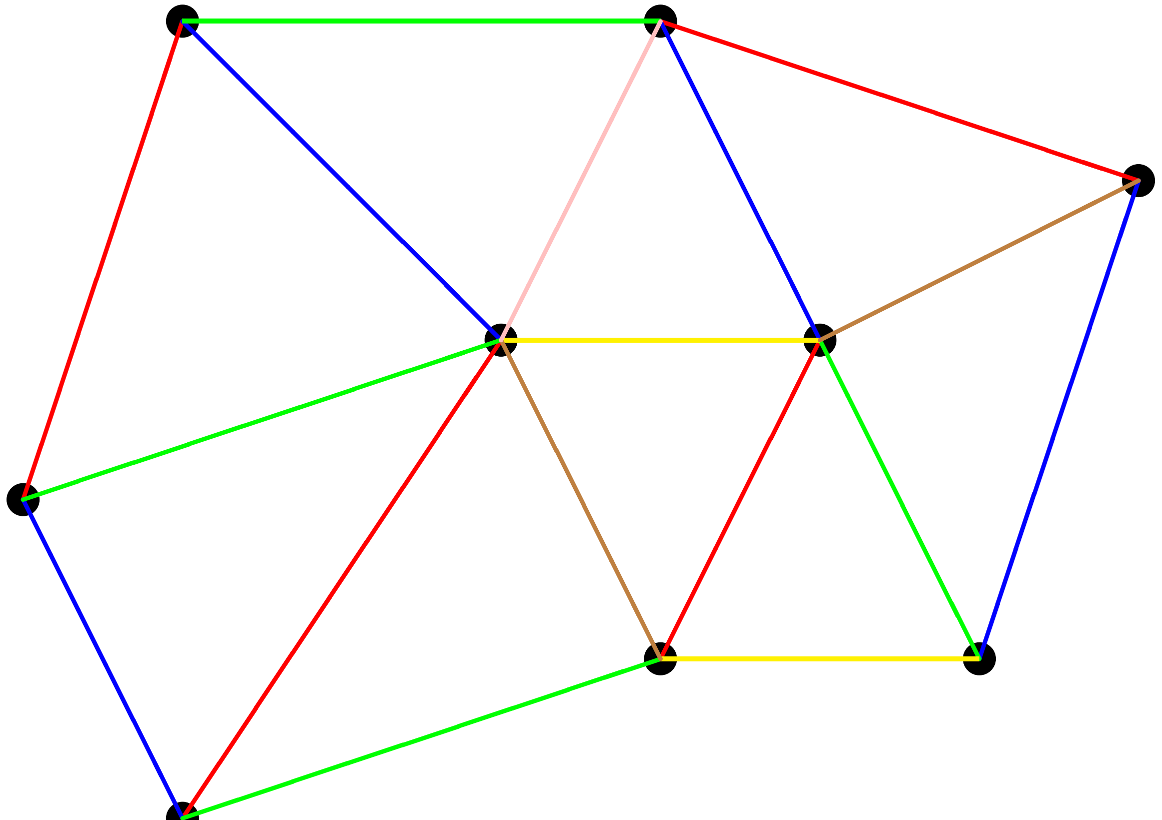
GPU Parallelisation

- block parallelism (50-1000)
  - on each GPU, data is broken into mini-partitions, worked on separately and in parallel by different SMs within the GPU
  - each mini-partition is sized so that all of the indirect data can be held in shared memory and re-used as needed
  - implementation requires re-numbering from global indices to local indices – tedious but not difficult
  - can use different mini-partitions for different parallel loops – “execution plan” generated during startup
- thread parallelism (32-128)
  - each mini-partition is worked on by a block of threads in parallel

Data dependencies

Data dependencies at thread block level avoided by “coloring” edges so no two edges of the same color update the same node at the same time

- parallel update for each color, then synchronize
- some loss of parallelism, but not too much



Airfoil test code

- 2D Euler equations, cell-centred finite volume method with scalar dissipation (miminal compute per memory reference – should consider switching to more compute-intensive “characteristic” smoothing more representative of real applications)
- roughly 1.5M edges, 0.75M cells
- 5 parallel loops:
  - `save_soln` (direct over cells)
  - `adt_calc` (indirect over cells)
  - `res_calc` (indirect over edges)
  - `bres_calc` (indirect over boundary edges)
  - `update` (direct over cells with RMS reduction)

Airfoil test code

Library is instrumented to give lots of diagnostic info:

```
new execution plan #1 for kernel res_calc
number of blocks            = 11240
number of block colors = 4
maximum block size          = 128
average thread colors  = 4.00
shared memory required = 3.72 KB
average data reuse          = 3.20
data transfer (used)    = 87.13 MB
data transfer (total)   = 143.06 MB
```

Airfoil test code

Single precision performance for 1000 iterations on an NVIDIA C2070:

count	time	GB/s	GB/s	kernel name	PS	BS
1000	0.22	101.8		save_soln		512
2000	1.09	74.1	75.4	adt_calc	256	128
2000	4.95	36.9	60.6	res_calc	128	128
2000	0.10	5.3	20.0	bres_calc	64	128
2000	1.03	94.7		update		64
TOTAL	7.40					

Max bandwidth is about 130GB/s, so this application is bandwidth limited.

Airfoil test code

Double precision performance for 1000 iterations on an NVIDIA C2070:

count	time	GB/s	GB/s	kernel name	PS	BS
1000	0.44	104.9		save_soln		512
2000	2.62	52.9	53.8	adt_calc	256	128
2000	10.35	30.5	50.8	res_calc	128	128
2000	0.08	11.2	27.9	bres_calc	64	128
2000	1.87	104.5		update		64
TOTAL	15.36					

Airfoil test code

Single precision performance on two Intel “Westmere” 6-core 2.67GHz X5650 CPUs using 16 OpenMP threads:

count	time	GB/s	GB/s	kernel name	PS
1000	1.68	13.7		save_soln	
2000	11.15	7.3	7.5	adt_calc	128
2000	16.57	10.3	11.2	res_calc	1024
2000	0.16	3.2	11.9	bres_calc	64
2000	4.67	20.9		update	
TOTAL	34.25				

Max bandwidth is about 25GB/s, so again bandwidth limited.

Airfoil test code

Double precision performance on two Intel “Westmere” 6-core 2.67GHz X5650 CPUs using 12 OpenMP threads:

count	time	GB/s	GB/s	kernel name	PS
1000	2.51	18.3		save_soln	
2000	11.68	11.8	11.9	adt_calc	1024
2000	20.99	12.8	13.5	res_calc	1024
2000	0.17	5.0	12.4	bres_calc	512
2000	9.29	21.1		update	
TOTAL	44.64				

Conclusions

- have created a high-level framework for parallel execution of unstructured grid algorithms on GPUs and other many-core architectures
- looks encouraging for providing ease-of-use, high performance and longevity through new back-ends
- auto-tuning is useful for code optimisation, and a new flexible auto-tuning system has been developed
- C2070 GPU speedup versus two 6-core Westmere CPUs is roughly $5\times$ in single precision, $3\times$ in double precision
- currently testing MPI version for CPU clusters; multi-GPU version should be running soon
- key challenge then is to build user community

Acknowledgements

- Carlo Bertolli, David Ham, Paul Kelly, Graham Markall and others (Imperial College)
- Nick Hills (Surrey) and Paul Crumpton (original OPlus development)
- Yoon Ho, Leigh Lapworth, David Radford (Rolls-Royce) Jamil Appa, Pierre Moinier (BAE Systems)
- Tom Bradley, Jon Cohen and others (NVIDIA)
- EPSRC, TSB, NVIDIA and Rolls-Royce for financial support
- Oxford Supercomputing Centre

Papers available from webpage:
`http://people.maths.ox.ac.uk/gilesm/op2/`