Imperial College
London

Imperial College London

Abstracting the hardware

Engineering climate models for the hardware revolution.

David A. Ham[1,2] Graham R. Markall[3] Florian Rathgeber[1]
Paul H.J. Kelly[3] Carlo Bertolli[3] Mike B. Giles[4] Gihan R. Mudalige[5]

[1]Department of Earth Science and Engineering, Imperial
[2]Grantham Institute for Climate Change, Imperial
[3]Department of Computing, Imperial
[4]Oxford-Man Institute of Quantitative Finance
[5]Oxford e-Research Centre

AMCG

The Free Lunch Is Over

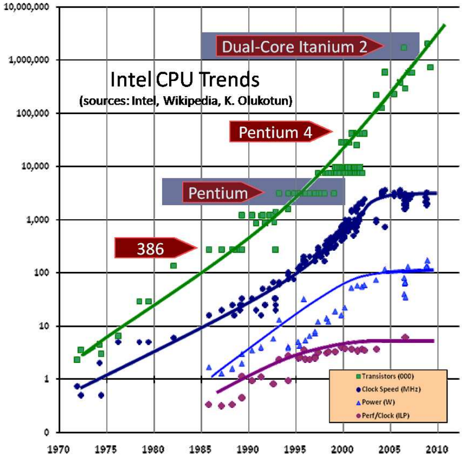


Image from Sutter, H., 2005, updated 2009. The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb s Journal 30 (3), 16–20

The future according to NVIDIA

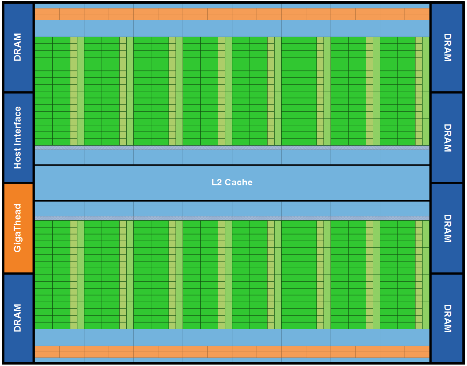


Image from Glasowsky, P. N., 2009. NVIDIA's fermi: The first complete GPU computing architecture. Tech. rep., NVIDIA

Imperial College London

The future according to NVIDIA

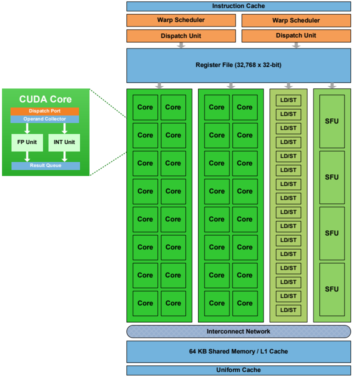
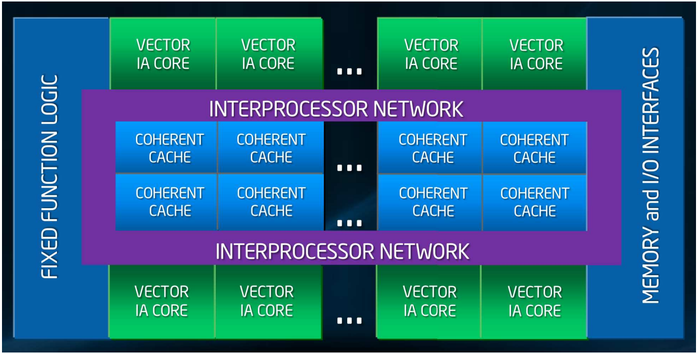


Image from Glasowsky, P. N., 2009. NVIDIA's fermi: The first complete GPU computing architecture. Tech. rep., NVIDIA

Imperial College London

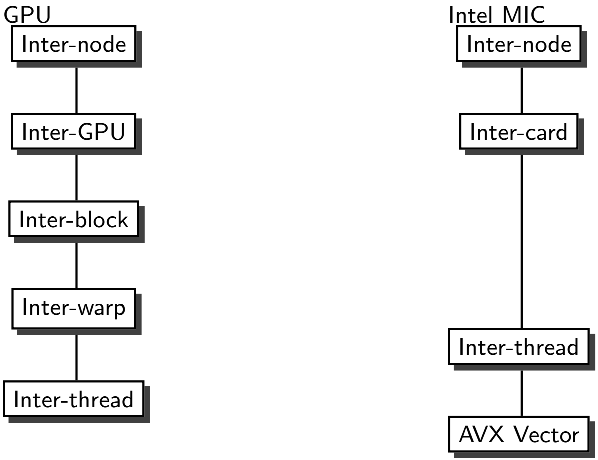
The future according to Intel



Schematic of the proposed Intel Many Integrated Core architecture.

Imperial College London

Different layers of paralellism.



Disruptive changes to the programming model.

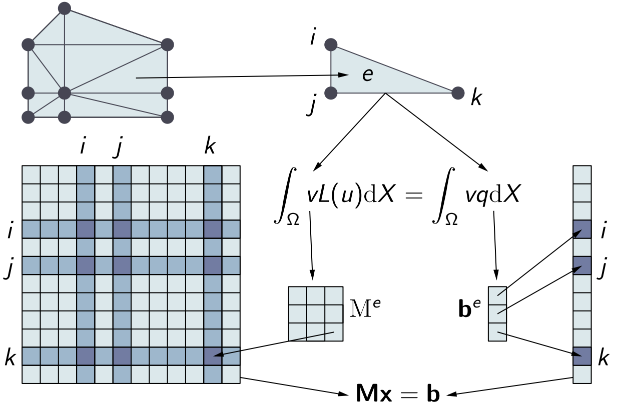
- (very) fine grain parallelism.
- multiple layers of parallelism.
- intrusive changes in low-level code.
- data layout changes necessary.
- compute is cheap, data is expensive.
- difficult to debug.

Disruptive changes to the programming model.

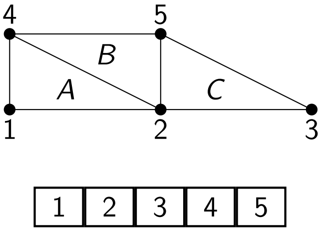
- (very) fine grain parallelism.
- multiple layers of parallelism.
- intrusive changes in low-level code.
- data layout changes necessary.
- compute is cheap, data is expensive.
- difficult to debug.

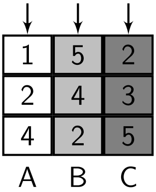
And what is optimal will change often!

Finite element assembly, a brief reminder

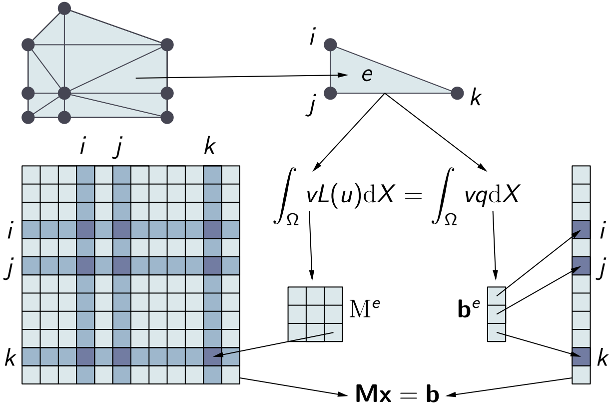


Disruptive code changes: memory layout





Disruptive changes: global assembly



Local matrix approach

We can write the global assembly operation as:

$$\mathbf{M} = \mathcal{A}^T \mathbf{M}^E \mathcal{A} \tag{1}$$

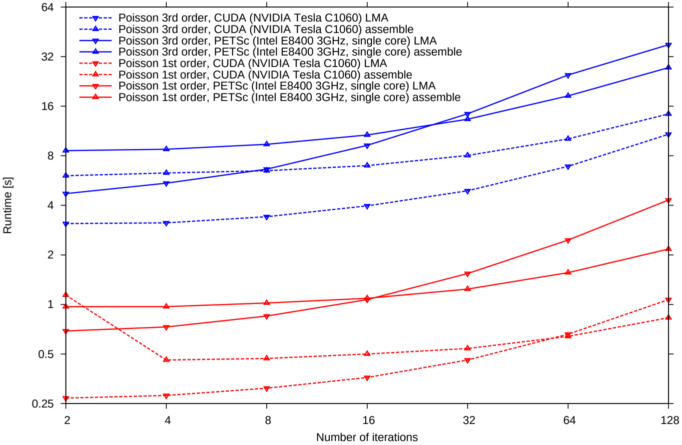
$$\mathbf{b} = \mathcal{A}^T \mathbf{b}^E \tag{2}$$

The local matrix approach consists of actually executing this sequence every time $\mathcal{A}\mathbf{v}$ is calculated:

$$\underbrace{\mathbf{t} = \mathcal{A}\mathbf{v}}_{\text{Stage 1}}, \qquad \underbrace{\mathbf{t}' = \mathbf{M}^e\mathbf{t}}_{\text{Stage 2}}, \qquad \underbrace{\mathbf{y} = \mathcal{A}^T\mathbf{t}'}_{\text{Stage 3}}.$$

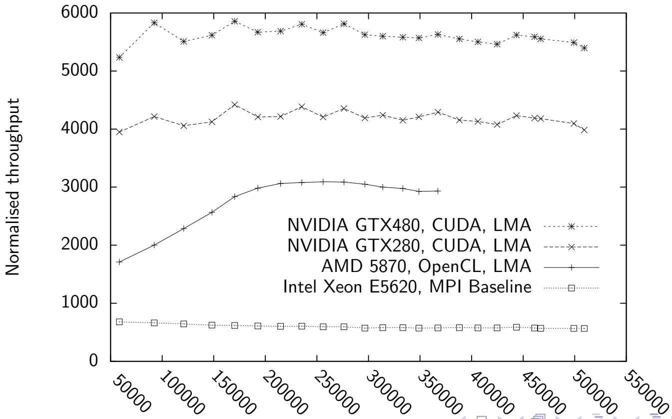
Cantwell, C. D., Sherwin, S. J., Kirby, R. M., Kelly, P. H. J., 2010. From h to p efficiently: strategy selection for operator evaluation on hexahedral and tetrahedral elements. Computers & Fluids

Trade-off between iteration count and method



The proof of the pudding

2D Advection diffusion equation.



The problem

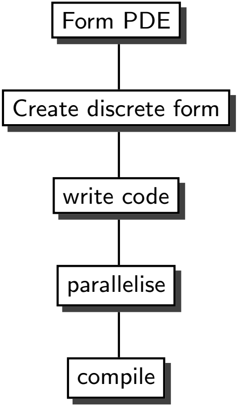
User Requirements

- Programmability: matching skills to tasks.
- Performance portability.
- Durability of programming effort.

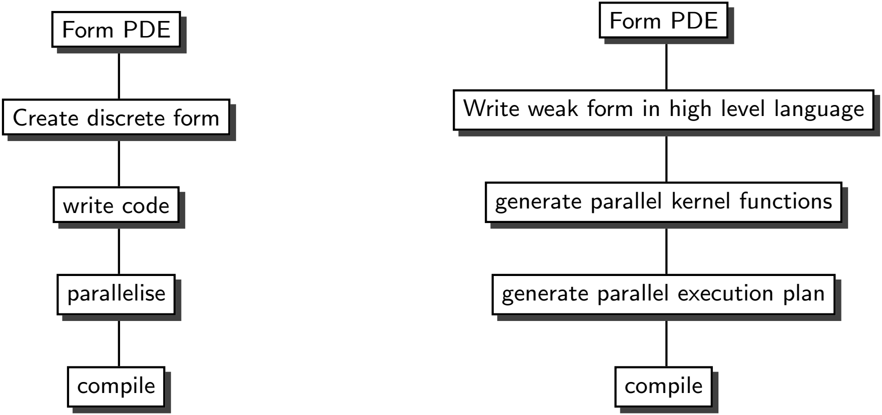
Reality of new platforms

- Hard to program. Developers must have huge skill sets.
- Performance is not portable.
- Constant changes in hardware invalidate previous effort.

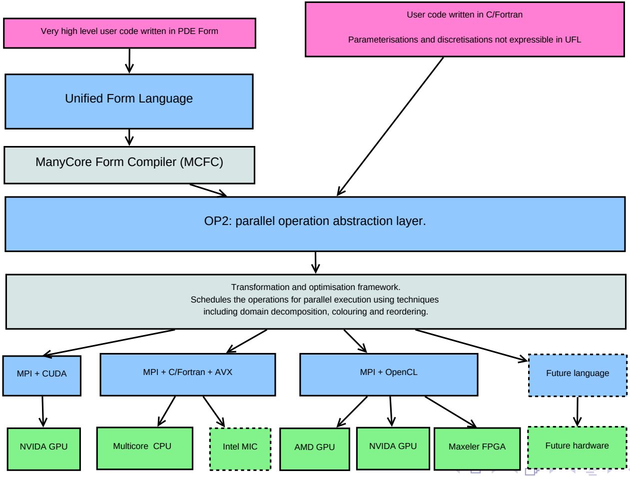
A solution: don't programme your model



A solution: don't programme your model



Multilayer abstractions for PDEs



Unified Form Language

Consider Poisson's equation in weak form:

$$\int_\Omega \nabla\phi \cdot \nabla\psi \mathrm{dx} = \int_\Omega \phi f \mathrm{dx}$$

```
phi=TestFunction(Psi)
psi=TrialFunction(Psi)
lhs=dot(grad(phi),grad(psi))*dx
rhs=phi*f*dx
Psi=solve(lhs,rhs)
```

UFL was developed by Marten Alnæs and Anders Logg for the FEniCS project.

Slightly less trivial example: C-grid linear shallow water equations

```
V = FunctionSpace(mesh, 'Raviart-Thomas', 1)
H = FunctionSpace(mesh, 'DG', 0)
W = V*H
(v, q) = TestFunctions(W)
(u, h) = TrialFunctions(W)
M_u = inner(v,u)*dx
M_h = q*h*dx
Ct = -inner(avg(u),jump(q,n))*dS
C = c**2*adjoint(Ct)
F = f*inner(v,as_vector([-u[1],u[0]]))*dx
A = assemble(M_u+M_h+0.5*dt*(C-Ct+F))
A_r = M_u+M_h-0.5*dt*(C-Ct+F)
```

Slightly less trivial example: C-grid linear shallow water equations

Maths as code. The divergence and pressure gradient:

$$\int_\Omega q \nabla \cdot \mathbf{u} \,\mathrm{d}V = -\int_{\Gamma E} \mathbf{u} \cdot \mathbf{n} (q^+ - q^-) \,\mathrm{d}S$$

$$c^2 \int_\Omega \mathbf{v} \cdot \nabla h \,\mathrm{d}V = c^2 \int_{\Gamma E} (h^+ - h^-) \mathbf{n} \cdot \mathbf{v} \,\mathrm{d}S$$

become:

```
Ct = -inner(avg(u),jump(q,n))*dS
C = c**2*adjoint(Ct)
```

Side note: UFL + libadjoint = automatic adjoints

- Automatically differentiating UFL code is as easy as formulating the continuous adjoint.
- Libadjoint facilitates adjoint models by recording the forward model via annotation the solution of linear systems in the source code.
- Using UFL these annotations can be automated.

Farrell, P. E. and Funke, S. W. and Ham, D. A. *A new approach for developing discrete adjoint models*, Submitted to ACM Transactions on Mathematical Software

The OP2 Layer

Kernel API

- Specify innermost loops such as integral over one (low order) element.
- Only aware of local assembly operations.
- Written in C++ or Fortran

Global API

- Specifies relationship of local problem to global.
- Specifies parallel operations without specifying order.

The OP2 API

Sample Kernel

```fortran
subroutine rhs_kernel(rhs,f,vol)
  ! Locally assembled RHS
  double precision, dimension(3), intent(inout) :: rhs
  ! RHS function
  double precision, dimension(3,2), intent(in) :: f
  ! Element volume
  double precision :: vol

  integer i,j

  !! Local actions in terms of f(i,j), rhs(i)

end subroutine rhs_kernel
```

The OP2 API

```
op_set :: elements, linear_dofs
op_dat :: rhs_dat, f_dat, vol_dat
op_map :: e2linear

!! Initialisation of set sizes and map data.

call op_par_loop(rhs_kernel, elements, &
     op_arg(rhs_dat, e2v, OP_ALL, OP_INC), &
     op_arg(f_dat, e2v, OP_ALL, OP_READ), &
     op_arg(vol_dat, OP_ID, -1, OP_READ))
```

What's actually happening?

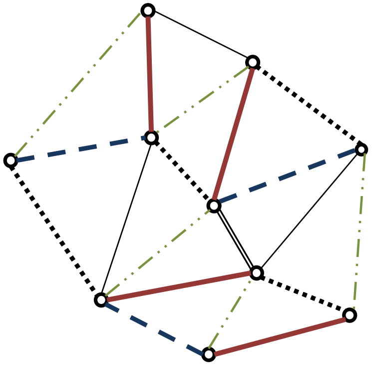
- Prototype codes and compilers work (and give results shown here).
- Significant effort at Imperial Computing and ESE, and Oxford to
  - produce the compiler from UFL to OP2
  - expand the OP2 prototype to full capacity, including MPI.
  - make OP2 ready to support Hydra (Rolls Royce Turbine code)
- Significant buy-in already from Rolls, EPSRC and NERC.
- Currently working on shallow water prototype for Gung-Ho
- BAe systems are also interested.

Overview

- The hardware landscape is changing, fast.
- Writing scientific software is labour-intensive, error-prone and not performance portable.
- This is far worse on emerging massively parallel architectures.
- Conventional software engineering prevents computational scientists and computer scientists working on the same problem.
- Code generation offers us a way out.

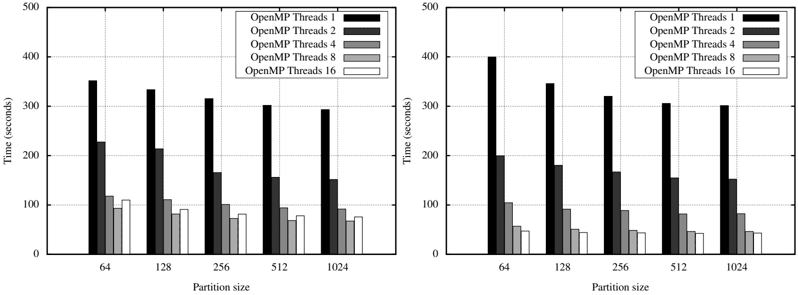
This research is supported by EPSRC, NERC, Rolls Royce, the Technology Strategies Board and the UK Met Office.
Giles, M., Mudalige, G., Sharif, Z., Markall, G., Kelly, P., 2011a. Performance analysis and optimization of the op2 framework on many-core architectures. The Computer Journal
Markall, G. R., Ham, D. A., Kelly, P. H. J., 2010. Towards generating optimised finite element solvers for gpus from high-level specifications. Procedia Computer Science 1 (1), 1815 – 1823, ICCS 2010
Giles, M., Mudalige, G., Sharif, Z., Markall, G., Kelly, P., 2011b. Performance Analysis of the OP2 Framework on Many-core Architectures. ACM SIGMETRICS Performance Evaluation Review

Two level colouring



Colouring of edges within one coloured partition.

Optimal CPU behaviour



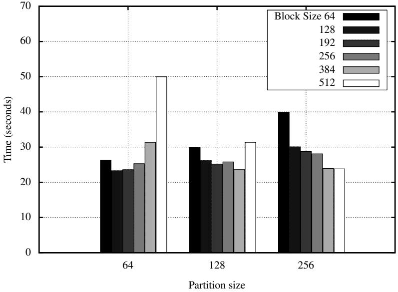
(a) Intel Xeon E5462 (Penryn)

(b) Intel Xeon E5540 (Nehalem)

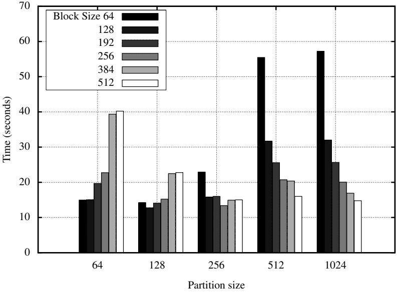
Flow past an airfoil: nonlinear inviscid flow with C grid finite volume discretisation.

Effect of changing partitioning and thread count on 8-core Intel machines of different processor generations.

Optimal GPU behaviour



(a) GTX260



(b) Tesla C2050

Flow past an airfoil: nonlinear inviscid flow with C grid finite volume discretisation.

Effect of changing partitioning and block size on GPUs of different processor generations.

The OP2 layer

```
op_par_loop_6(adt_calc,"adt_calc",cells,
              p_x,    0,pcell,  2,"float",OP_READ,
              p_x,    1,pcell,  2,"float",OP_READ,
              p_x,    2,pcell,  2,"float",OP_READ,
              p_x,    3,pcell,  2,"float",OP_READ,
              p_q,  -1,OP_ID,  4,"float",OP_READ,
              p_adt,-1,OP_ID,  1,"float",OP_WRITE);
```

Conclusions

- The hardware landscape is changing, fast.
- Writing scientific software is labour-intensive, error-prone and not performance portable.
- Code generation offers us a way out.