Automated Scientific Computing

CMSC 34900 Topics in Scientific Computing

L. Ridgway Scott (ridg@uchicago.edu)

The course will involve reading papers in the field and making class presentations.

The presentations will involve both papers of others and also on-going research that is not yet published.

Some of the class work will cover research done in the FEniCS project and the Spiral Project.

New paradigms for software development

Hierarchical code development

Improves upon standard two level model: language+compiler

Allows appropriate optimizations to be done at different levels

Improves code correctness and decreases cost of code development

Automation of code production

Utilizes abstract descriptions as basis to generate code automatically

Leverages intrinsic domain languages

Improves code correctness and decreases cost of code development

Allows true optimization of generated code

Hierarchy of problem representation

—Model description [application domain]

—Algorithm discovery [mathematical description]

—Algorithm implementation [programming language]

. . . . . .

—Executable code [machine code: parallel, multicore]

Automation can be performed at each level

Interaction between levels can be tested to optimize performance

Example of hierarchy: signal processing (Spiral project)

—Model description [Discrete Fourier Transform: DFT]

—Algorithm discovery [Cooley-Tukey FFT]

—Algorithm implementation [FFTW]

—Executable code [BLAS, ATLAS, OSKI]

Automation can be performed at each level: can derive FFT from abstract definition of DFT

No need to hand-code for specific architectures

Paradigm limitations?

Must have hierarchical structure

But what doesn't?

Common in computer architecture, compiler design, scientific computing, etc.

Automatic generation requires abstract model

In principle, this can be developed in any area, e.g., operating systems

Found in VLSI design, compiler design [3, 2], scientific computing [8, 7, 5, 4], etc.

Why focus on scientific computation?

Has built-in hierarchical structure

Models often built from simpler models

Laplace $\Longrightarrow$ Stokes $\Longrightarrow$ Navier-Stokes $\Longrightarrow$ Grade-2 fluids

Has built-in abstract descriptions

Equations are the language of science and engineering

$$E = mc^2$$

$$F = ma$$

$$-\nu\,\Delta\,\mathbf{u} + \mathbf{curl}(\mathbf{u} - \alpha\,\Delta\,\mathbf{u}) \times \mathbf{u} + \nabla\,p = \mathbf{f}$$

The software challenge

(Correct) interpretation of problem discription

Have to translate from a high-level description to low-level executable, correctly!

Optimization of generated code

Have to solve disparate optimization problems at compile time or in conjunction with run-time indicators

Tension between expressiveness and efficiency

Ideally the naive user could code in a high-level specification language and have it translated into efficient machine code.

Problem with single-language approach

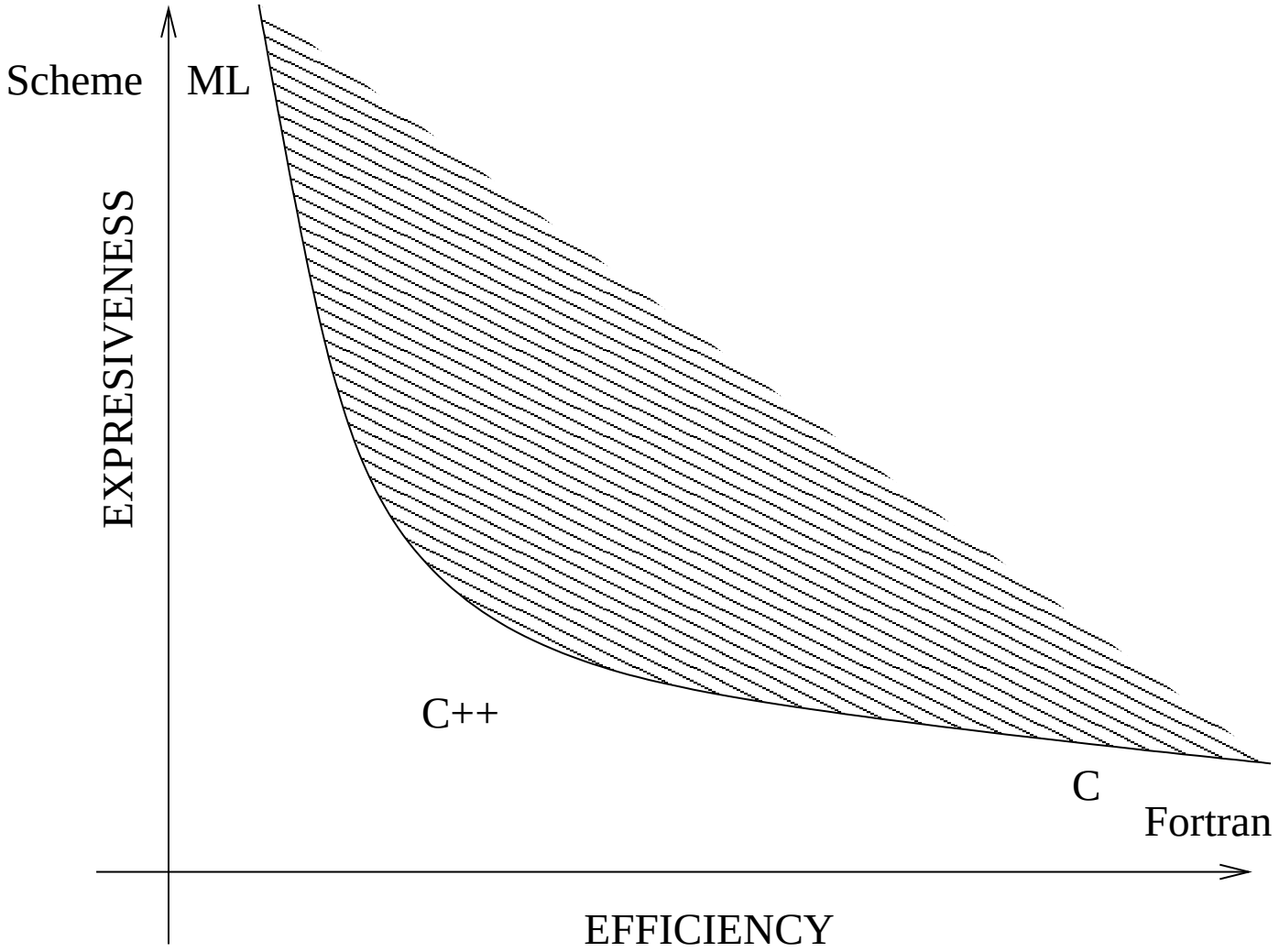

Figure 1: The conflict between efficiency and expressiveness [M. Dragichescu].

Causes of efficiency–expressiveness conflict

Compilers perform transformations, not optimization.

Optimization at compile time, even if based on realistic performance models, would be wildly expensive (NP^{NP})

True optimization would have to involve evaluating performance.

Actual performance is often data dependent.

Best transformations are often domain dependent.

What is missing currently

Current language support for hierarchical abstraction

- You can define what you want
- But you can't specify how to compile it

Compiler optimizations cannot be chosen to fit the application.

Must live with what the compiler writer gives you.

Interpretation of multiple abstraction levels may also be costly.

One solution: multi-lingual

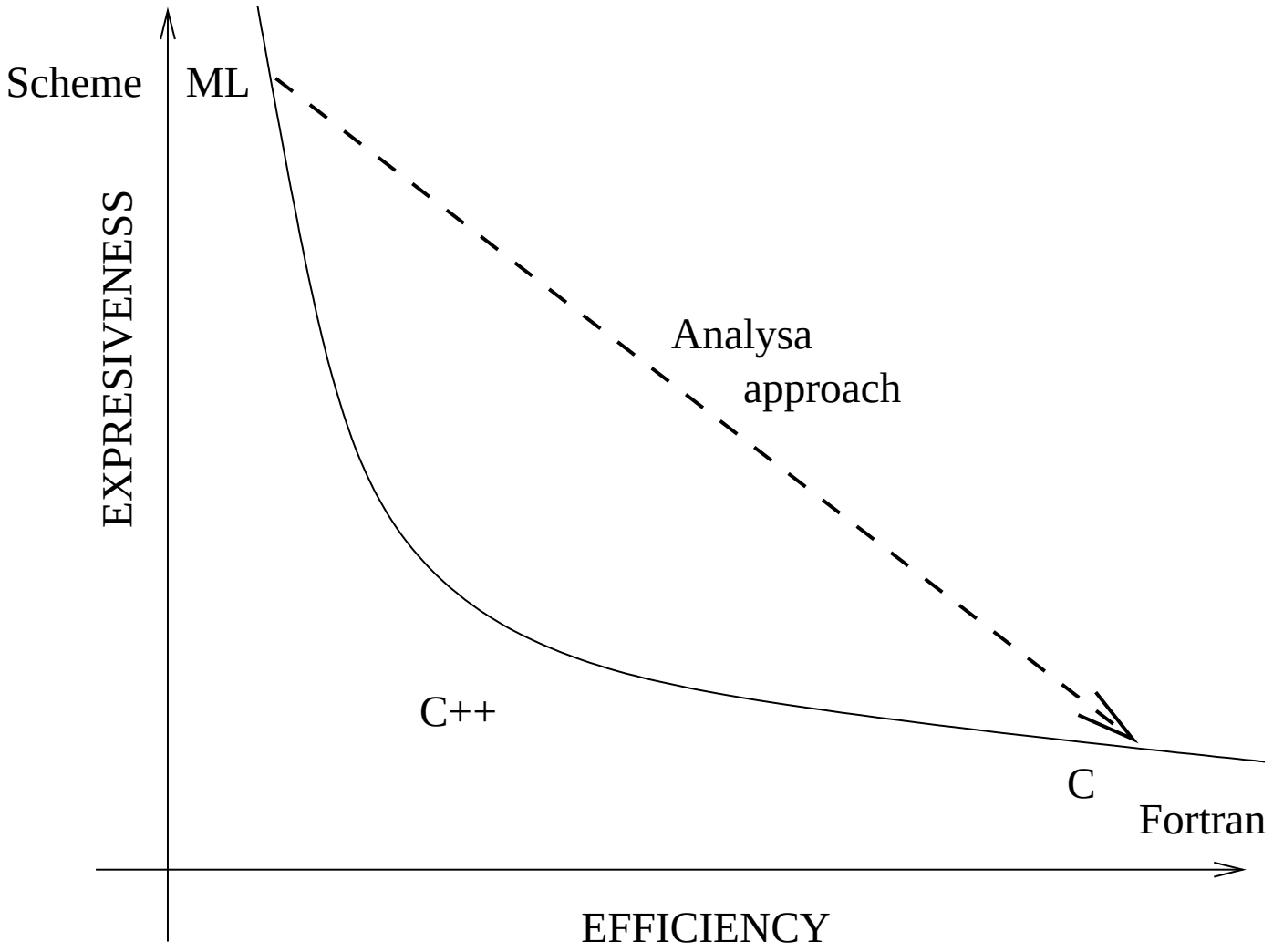

Figure 2: One way to combine efficiency and expressiveness [1].

Limits to linking multiple languages

(Experience gained from the development of Analysa.)

Lacks formal description, requires development (and maintenance) of ad hoc interface.

Different memory models (allocation, garbage collection) can interact adversely at run time.

Increases the burden of code maintenance (must track multiple standards, together with interactions between them).

But it does mollify the tension between expressiveness and efficiency.

Similar challenges

from which we can learn

Same as problems in language/compiler design

Need to perform optimization and code generation

Often need to utilize these in an application-specific way (Spiral).

Hierarchical issues addressed in programming language

Manticore project for parallel computation

Algegra of code optimizations

Common to use Lambda Calculus or algebra of real numbers

Finite elements introduces optimization in vector spaces

Course objectives

Read papers in the field (each person to make a presentation in class)

Perform a project involving automation, make a presentation to the class, and write-up a report

Develop an annotated bibliography: each person invited to contribute

Annotated bibliography

Want short description of each paper, together with some sort of organization into similar areas.

1 Efficiency–expressiveness trade-off

Analysa [1] combined efficiency and expressiveness by using a functional programming language (AlScheme) as a scripting language which linked with C, C++ and Fortran code for efficiency.

2 Compilers

Automatic generation of code has been performed in compiler design [3, 2].

3 Scientific computing

[8, 7, 5, 4], etc.

4 Algebra of compiler optimization

In [6], the authors (according to their abstract) “use Kleene algebra with tests to verify a wide assortment of common compiler optimizations, including dead code elimination, common subexpression elimination, copy propagation, loop hoisting, induction variable elimination, instruction scheduling, algebraic simplification, loop unrolling, elimination of redundant instructions, array bounds check elimination, and introduction of sentinels. In each of these cases, we give a formal equational proof of the correctness of the optimizing transformation.”

This will be posted on the course web site and up-dated based on class input.

A simple project idea: molecular dynamics (MD)

$$F = ma$$

means

$$\frac{d^2x}{dt^2} = \frac{1}{m}F(x)$$

but we usually think in terms of energy, not force

$$F = \nabla V$$

where V is the energy potential, such as Lennard-Jones

$$V(x) = \sum_{j \neq i} |x_i - x_j|^{-12} - |x_i - x_j|^{-6}$$

Also requires a confinement force to avoid drift to ∞.

Time stepping in MD

The code for the time-stepping scheme is not so bad.

The scheme is

$$U \leftarrow U + \Delta t\, F$$

$$X \leftarrow X + \Delta t\, U$$

```
function [xx,yy,zz,uu,vv,ww]=
              timestep(xx,yy,zz,uu,vv,ww,dt,kstep,rcut)
xs=xx; ys=yy; zs=zz;
for i=1:kstep
   [cfx, cfy,cfz,kb,remax]=confin(xx,yy,zz,rcut,1.0);
   [fx, fy,fz,amen,potl]=vecljpot(xx,yy,zz,rcut);
   fx=cfx+fx; fy=cfy+fy; fz=cfz+fz;
   uu=uu+dt*fx; vv=vv+dt*fy; ww=ww+dt*fz;
   xx=xx+dt*uu; yy=yy+dt*vv; zz=zz+dt*ww;
end
```

Computing the force in MD ...

```
function [fxl,fyl,fzl,amen,ljpot]=
                              vecljpot(xl,yl,zl,rcutsq)
n=max(size(xl));
amen=rcutsq;
ljpot=0;
fxl=zeros(1,n); fyl=zeros(1,n); fzl=zeros(1,n);
for i=2:n
   xli=xl-shift(xl,i-1);
   yli=yl-shift(yl,i-1);
   zli=zl-shift(zl,i-1);
   rs(i:n)=(xli(i:n).*xli(i:n))+
           (yli(i:n).*yli(i:n))+(zli(i:n).*zli(i:n));
   amen=min(min(rs(i:n)),amen);
      rsi(i:n)=1./rs(i:n);
      rfor(i:n)=rsi(i:n).*rsi(i:n);
      rsix(i:n)=rsi(i:n).*rfor(i:n);
      rate(i:n)=rfor(i:n).*rfor(i:n);
```

```
        rtlv(i:n)=rfor(i:n).*rate(i:n);
        ljpot=ljpot+sum(rtlv(i:n))-sum(rsix(i:n));
        rfrten(i:n)=rsi(i:n).*rfor(i:n).*rate(i:n);
        kfr(i:n) = 24*(2*rfrten(i:n)-rate(i:n));
        dfx(i:n) = kfr(i:n).*xli(i:n);
        dfy(i:n) = kfr(i:n).*yli(i:n);
        dfz(i:n) = kfr(i:n).*zli(i:n);
        fxl(i:n)=fxl(i:n)+dfx(i:n);
        fyl(i:n)=fyl(i:n)+dfy(i:n);
        fzl(i:n)=fzl(i:n)+dfz(i:n);
        fxl(1:n-i+1)=fxl(1:n-i+1)-dfx(i:n);
        fyl(1:n-i+1)=fyl(1:n-i+1)-dfy(i:n);
        fzl(1:n-i+1)=fzl(1:n-i+1)-dfz(i:n);
end
ljpot=8*ljpot;
```

... looks like a dog's breakfast

And we still need to compute the containment force

```
function [fxl,fyl,fzl,kb,remus]=
                    confin(xl,yl,zl,rfurw,stren)
n=max(size(xl)); kb=0; remus=rfurw;
for i=1:n
   are= (xl(i))**2 + (yl(i))**2 + (zl(i))**2 ;
   remus=max(are,remus);
   if are<rfurw
      fxl(i)=0; fyl(i)=0; fzl(i)=0;
   else
      kb=kb+1;
      are=are-rfurw;
      are=stren*are/(1+sqrt(are)); % subharmonic
fxl(i)=-xl(i)*are; fyl(i)=-yl(i)*are; fzl(i)=-zl(i)*are;
   endif
end
```

Wouldn't it be nice if ....

we had a system in which we could specify the force computation more mathematically

(more compact notation: $r^{-6} - r^{-12}$)

we could easily make changes in the force definitions to see how this changes the physics of the simulation

If we have to write it by hand, could we verify the force computation to be sure it does what we intend:

is it really r^{-6}? r^{-12}?

Tentative schedule

Tu 25 Sep	Course Intro [today]
Th 27 Sep	FEM intro
Tu 2 Oct	Review of FEM
Th 4 Oct	FErari
Tu 9 Oct	Review of FErari
Th 11 Oct	Claes Johnson
Tu 16 Oct	Review of ...

References

[1] Babak Bagheri and L. R. Scott. About Analysa. Research Report UC/CS TR-2004-09, Dept. Comp. Sci., Univ. Chicago, 2004.

[2] Christopher Warwick Fraser. *Automatic generation of code generators.* PhD thesis, 1977.

[3] Kingshuk Karuri. *A Framework for Automatic Generation of Code Optimizers*. PhD thesis, 2001.

[4] J. Korelc. Multi-language and multi-environment generation of nonlinear finite element codes. *Engineering with Computers*, 18:312–327, Nov 2002. 10.1007/s003660200028.

[5] Joze Korelc. Automatic generation of finite-element code by simultaneous optimization of expressions. *Theoretical Computer Science*, 187:231–248, Nov 1997.

[6] Dexter Kozen and Maria-Christina Patron. Certification of compiler optimizations using kleene algebra with tests. In *CL '00: Proceedings of the*

First International Conference on Computational Logic, pages 568–582, London, UK, 2000. Springer-Verlag.

[7] Robert van Engelen, Lex Wolters, and Gerard Cats. CTADEL: a generator of multi-platform high performance codes for PDE-based scientific applications. In *ICS '96: Proceedings of the 10th international conference on Supercomputing*, pages 86–93, New York, NY, USA, 1996. ACM Press.

[8] Paul S. Wang, Hui-Qian Tan, Atef F. Saleeb, and Tse-Yung P. Chang. Code generation for hybrid mixed mode formulation in finite element analysis. In *SYMSAC '86: Proceedings of the fifth ACM symposium on Symbolic and algebraic computation*, pages 45–52, New York, NY, USA, 1986. ACM Press.