CS601: Software Development for Scientific Computing

Autumn 2024

18/11/2024 : Summary

CS601 - Lecture Topics Covered

1-3 Introduction to Scientific Computing, Computational thinking, IEEE 754 floating point system (terminology, catastrophic cancellation, rounding off modes)	18-21 object orientation, OO programming in C++ (const and references, overloading and overriding, virtual functions, templates, STL, move operator)
4-6 Program development environment (compiler tool chain, conditional compilation, program layout in memory), Makefile	22-25 Structured grids (Discretization, error estimates, Finite difference method (FDM), stencil computation) 26-29 Unstructured grid (Finite Element Method (FEM)) 30 OO based design for solving on grid
7-13 Motifs-dense and sparse matrix computation (costs involved, computational intensity, introduction to BLAS and LAPACK APIs, gaxpy with banded matrices), vectorizing	31-33 – Fast Fourier Transforms and faster matvec using separable matrices 34-39 – N-body methods (All-pairs, BH, FMM), Metric trees. Dynamic Programming (DP) problems and their categorization.
14-17 tools for debugging, profiling, and documentation (gdb, valgrind, gprof, doxygen)	40-summary and revision

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

IEEE754 Floating Point System

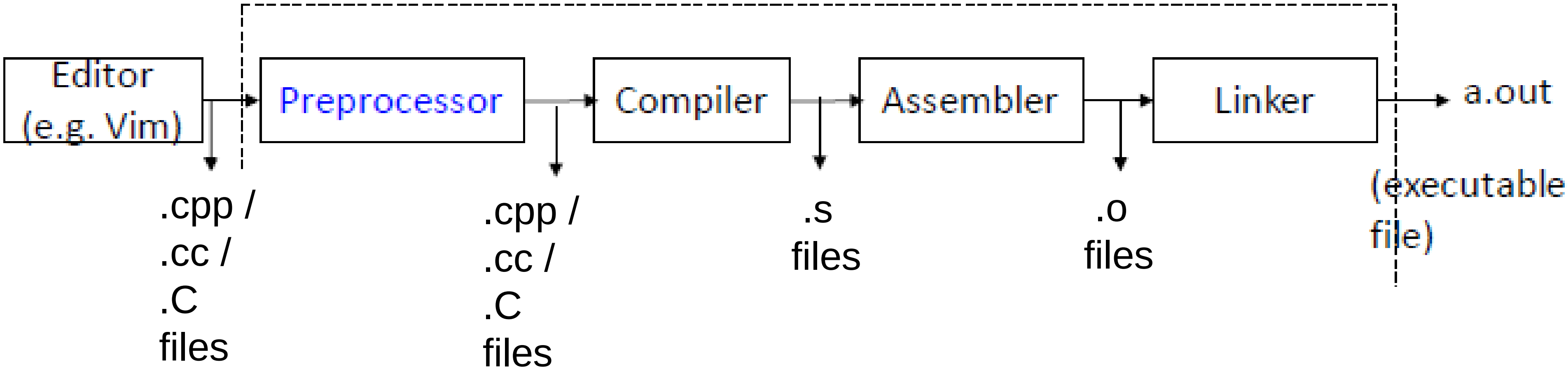
- Why this code is not robust Software Engineering?

```
double ComputeHypotenuse(double x, double y) {
    return sqrt(x*x + y*y);
}
```

Think: catastrophic cancellation

Program Development Environment

- `g++ 4_8_1.cpp -lm`



  - `g++` is a command to translate your source code (by invoking a collection of tools)
    - Above command produces `a.out` from `.cpp` file
  - `-l` option tells the linker to 'link' the math library

`Makefile` or `makefile`

- Is a file, contains instructions for the `make` program to generate a *target* (executable).
- Generating a target involves:
  1. Preprocessing (e.g. strips comments, conditional compilation etc.)
  2. Compiling (.c -> .s files, .s -> .o files)
  3. Linking (e.g. making printf available)
- A `Makefile` typically contains directives/instructions on how to do steps 1, 2, and 3.

Matrix Multiplication Performance

C=C+A*B, Square matrices, Dimensions = 2048x2048 (`INPUT_SIZE = 2048`)

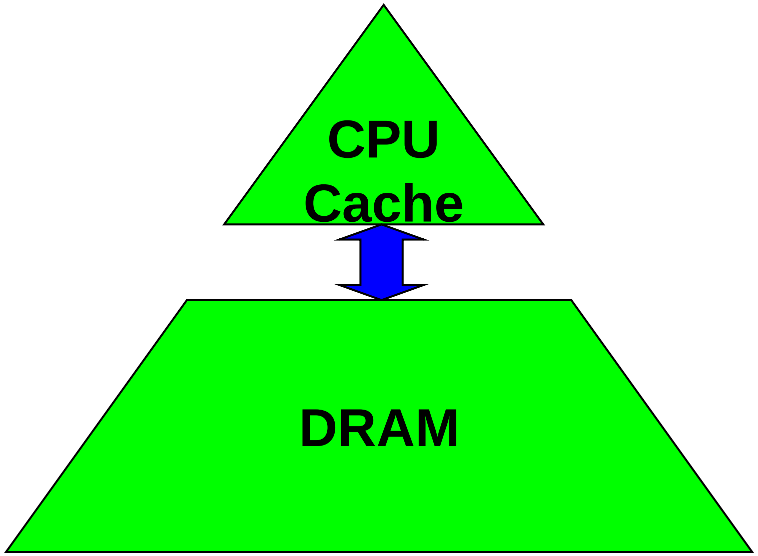
	Execution Time	**Speedup (w.r.t. Python)**
Python	2088.75s	1.0
C++	92.7s	22.53
+ –O3	41.67s	50.13
+ ikj loop ordering	4.71s	443.47
+ utilizing all cores (parallel)	0.147s	14209.18

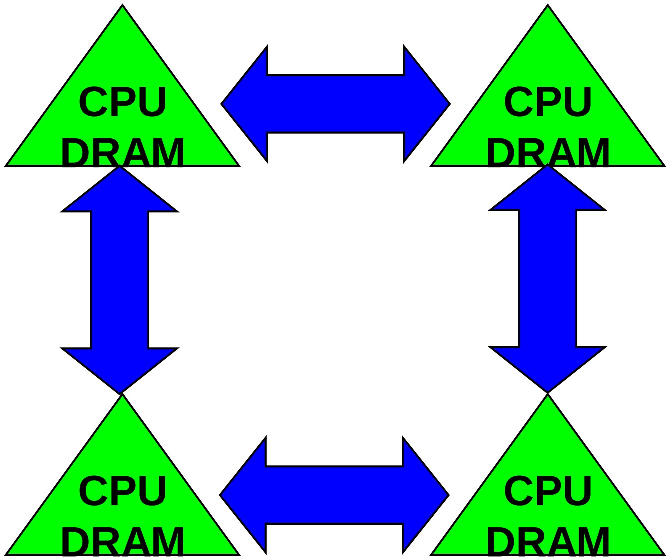
Also saw other kernels: Dot Product, AXPY, Matvec

Costs Involved

1. Arithmetic (FLOPS)
2. Communication: moving data between
   – levels of a memory hierarchy (sequential case)
   – processors over a network (parallel case).

matmul_blocked, matmul_recursive, matmul with different loop orderings, Sparse matrices – diagonal, banded. Matvec with sparse matrices.

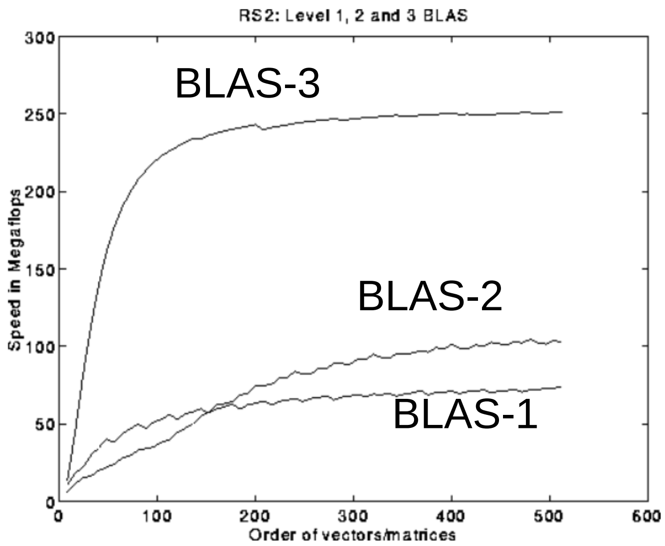




BLAS – Basic Linear Algebra Subroutines

- Level-1 or BLAS-1 (46 operations, routines operating on vectors mostly)
  - axpy, dot product, rotation, scale, etc.
  - 4 versions each: **S**ingle-precision, **d**ouble-precision, **c**omplex, complex-double (**z**)
  - E.g. saxpy, daxpy, caxpy etc.
  - **Do O(n) operations on O(n) data.**
- Level-2 or BLAS-2 (25 operations, routines operating on matrix-vectors mostly)
  - E.g. GEMV ($\alpha A.x + \beta y$), GER (Rank-1 update $A = A + y.x^T$), Triangular solve ($y = T.x, T\ is\ a\ triangular\ matrix$) etc.
  - 4 versions each, **do O(n^2) operations on O(n^2) data.**
- Level-3 or BLAS-3 (9 basic operations, routines operating on matrix-matrix mostly)
  - GEMM ($C = \alpha A.B + \beta C$),
  - Multiple triangular solve ($Y = TX$, T is triangular, X is rectangular)
  - **Do O(n^3) operations on O(n^2) data.**

- ***Why categorize as BLAS-1, BLAS-2, BLAS-3? Performance***



source: http://people.eecs.berkeley.edu/~demmel/cs267/lecture02.html

Tools

- Valgrind
  - Memcheck
  - Cachegrind
- Gprof
- Doxygen
- GDB

Need for returning references- Example1

- How can we assign one object to another?

```
Apple a1("Apple", 1.2); //constructor Apple::Apple(string, float)
                          //is invoked
Apple a2; //constructor Apple::Apple() is invoked.
a2  = a1 //object a1 is assigned to a2;assignment operator is invoked
```

```
Apple& Apple::operator=(const Apple& rhs)
```

Called Copy Assignment Operator

```
Apple& Apple::operator=(const Apple& rhs) {
commonName = rhs.commonName;
weight = rhs.weight;
energyPerUnitWeight = rhs.energyPerUnitWeight;
constituents = rhs.constituents;
return *this;
}
```

What is Move Assignment Operator?

Other topics in C++

- Overloading and overriding, OO programming
- Function templates
- Class templates
- STL

```
  double dResult=0.;
  dResult = cs601::scprod(dim, vector1, vector2);
int iResult=0.;
//multiply vector of int and store the result in a new int
iResult = cs601::scprod(dim, vector3, vector4);
```

Grids and Computation on Grids

- FDM
  - Explicit method, Implicit method (1D problem, time domain)
  - 2D problem (no time domain)
- FEM
  - 1D problem

  [Multi-scale Multi-physics Heart Simulator UT-Heart]

- Design solution using classes

Divide-and-Conquer FFT (D&C FFT)

FFT(v, ϖ, m) … assume m is a power of 2

if m = 1 return v[0]

else

v_{even} = FFT(v[0:2:m-2], ϖ^2, m/2)

v_{odd} = FFT(v[1:2:m-1], ϖ^2, m/2)

ϖ-vec = [ϖ^0, ϖ^1, … $\varpi^{(m/2-1)}$] ← precomputed

return [v_{even} + (ϖ-vec .* v_{odd}),

v_{even} - (ϖ-vec .* v_{odd})]

- Matlab notation: ".*" means component-wise multiply.

Cost: T(m) = 2T(m/2)+O(m) = O(m log m) operations.

Popularized/published by Cooley-Tuckey in 1965.

https://www.math.uci.edu/~chenlong/MathPKU/FMMsimple.pdf

1. Separable and low rank matrices

Consider the following simple example of (1) with

$$\phi_{ij} = (x_j - y_i)^2 = y_i^2 - 2x_j y_i + x_j^2,$$

where $\boldsymbol{x} = (x_j) \in \mathbb{R}^N, \boldsymbol{y} = (y_i) \in \mathbb{R}^N$ are two given vectors. Then, for $i = 1, \cdots, N$

$$u_i = \left(\sum_{j=1}^{N} q_j\right) y_i^2 - 2\left(\sum_{j=1}^{N} q_j x_j\right) y_i + \left(\sum_{j=1}^{N} q_j x_j^2\right) = \alpha y_i^2 - 2\beta y_i + \gamma. \tag{3}$$

The coefficients α, β, and γ do not depend on i and can be computed in $\mathcal{O}(N)$ operations for one pass. Then another loop can compute the summation in $\mathcal{O}(N)$ operations. We list the $\mathcal{O}(N^2)$ naive summation algorithm in `summation1` and an $\mathcal{O}(N)$ algorithm in `summation2`. In `summation2`, the nested for loops are separated. This is a simple example of separable matrices.

1

N-Body Methods

- All-pairs
- Barnes-Hut
  - Metric trees
- FMM

Dynamic Programming (DP)

- 1D problem
- Gap problem
- Parenthesis problem
- RNA problem

Concluding Thoughts

"The future isn't only in computer science. Computer science can be key to building many futures." - Mark Guzdial, Professor of EECS, Michigan State Univ.

(from blog on creating elite engineers)
https://cacm.acm.org/blogs/blog-cacm/254883-the-role-of-computer-science-in-elite-higher-education-seeing-the-expert-blind-spot/fulltext

- The world rewards initiative.

- The world rewards risk takers.
  - Don't worry about failures

Concluding Thoughts