PyGMG: A High Performance, Portable and Expressive Implementation of Multigrid in Python

Shiv Sundram

ASPIRE is a Research Lab at the University of California, Berkeley for computer architecture/high performance computing with joint projects in collaboration with Lawrence Berkeley National Laboratory. https://aspire.eecs.berkeley.edu/about/ I am a student researcher here since December 2014, employed part time by the Lawrence Berkeley Laboratory. I work under the guidance of Armando Fox , UC Berkeley EECS Prof & co-PI ASPIRE Lab, and Kathy Yelick, UCB EECS Prof and LBL Associate Director of Computing Sciences.

This is a poster I presented at the annual ASPIRE lab conference in Santa Cruz, CA, based on my work in the first half of 2015. In this poster, we discuss an implementation of multigrid algorithms that is portable (architecture independent), that can achieve high performance on multicore CPUs and GPUs, and yet is written sequentially such that all parallel code is abstracted away from the developer.

The original implementation of this algorithm, on the other hand, is used widely in the supercomputing community but is verbose and has to be rewritten for different architectures. Our implementation on the other hand, utilizes a code generation platform that allows us to generate code for a pre-specified platform. This allows the maintenance of one easily readable code base but still have the ability to target various architectures.

A follow up presentation is planned for the next ASPIRE conference in Lake Tahoe in early 2016, in which I will discuss improvements to the project. In addition, in a separate project inspired by this work, my colleagues will be presenting their work on a new Domain Specific Language for stencil computations that utilizes code generation, and how it can be used to enhance this software and parallel software in general.

PyGMG: A SEJITS APPLICATION

A High Performance, Portable, and Expressive Implementation of Multigrid in Python

Shiv Sundram, Nathan Zhang, Chick Markley, Sam Williams, Kathy Yelick, Armando Fox

Problem

Multigrid is numerical technique used to efficiently solve elliptic PDE's (eg Helmholtz, Poisson). Mathematically, this is equivalent to solving a particular sparse matrix equation in O(N) time (where N is the size of the matrix). The existing C implementation (HPGMG) from LBNL is a finite-volume algorithm which is efficient but extremely verbose. Furthermore, it is heavily hardcoded/ optimized for a very specific platform (not portable)

Motivation for Python Implementation

3 Goals: **Portability**, **Efficiency**, **Readability**

- **Portability**: Should be able to use different platforms (supercomputer, GPU, multicore CPU, or any combination)using one code base (using SEJITS to generate code for each platform)
- **Efficiency**: Should be as fast as HPGMG, and faster to develop/improve
- **Readability**: parallelization of code should be abstracted away from developers and users

Implementation details

- Sequence of stencil computations on a grid
- Stencils/Operators (eg smoothers, interpolators) are modular and adhere to a simple interface.
- Build various data structures wrapped around tuples and numpy arrays to represent grid
- Smoothers: Weighted Jacobi and Chebyshev
- Bottom Solvers: BigCStab, or just repeated smoothing
- Boundary Conditions: Supports both Periodic and Dirichlet

Figure 1: Visualization of stencil computation
Source: http://en.wikipedia.org/wiki/Stencil_code

Algorithm: V-cycle

Multigrid can be decomposed into a series of 3D stencil computations called V-Cycles

- Interpolation/restriction of grid x
- Residual & sparse/symmetric matrix multiply
- Iterative solving of Ax=b (smoothing)

```
Function MGV ( b(i), x(i) )
  //Multigrid V-Cycle
  //Solve T(i)*x(i) = b(i)
  //given b(i) & initial guess for x(i)
  //return improved x(i)
  //let i=log2(size(side(grid))

  if (i = 1)
    //compute exact solution x(1) of P(1) only 1 unknown
    return x(1)
  else
    //smooth/improve solution
    x(i) = Smooth (b(i), x(i))
    r(i) = T(i)*x(i) - b(i) //residual

    r(i-1) = Restrict(r(i))
    x(i-1) = MGV(r(i-1), 0) //recursive call
    d(i)   = Interpolate(x(i-1))

    x(i) = x(i) - d(i) //correct fine grid solution
    x(i) = Smooth(b(i), x(i)) //smooth again
    return x(i)
```

(Algorithm form Jim Demmel's CS267 Lecture Slides)

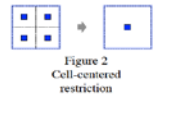

Figure 2
Cell-centered restriction

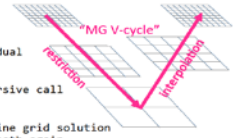

Figure 3
V-cycle

Algorithm: F-cycle

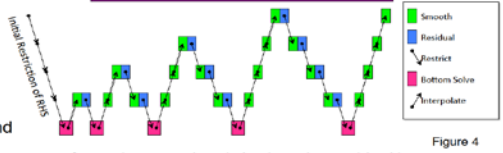

Figure 4

Can reduce error in solution by at least 1 bit with each call to an F-Cycle, which is simply a sequence of chained V-cycles of increasing initial grid precision

Results: HPGMG vs PyGMG

HPGMG	PyGMG
Hardcoded to only solve 3D grid	Can operate on grids of an arbitrary # of dimensions
MPI and OpenMP scattered throughout code	All code is sequential and thus clearly descriptive of algorithm. No MPI or OpenMP
Can only run code on CPUS & supercomputers	Can theoretically use SEJITs to run code on any computational platform
Equations for generating inputs to grid must be differentiated/manipulated separately and then written into code	Manipulation and differentiation done automatically by Sympy, a computer algebra solver
High Performance	High Performance once coupled with SEJITS

Future Work

Next Steps

- Code generation using SEJITS to achieve Portability
- Bechmark SEJITS generated code against HPGMG to show Efficiency

Other possible areas of interest

- Applying Z-order indexing to grid for increased cache locality
- Overload operators (MATLAB style syntax) instead of using function calls for manipulating grids to achieve even greater Readability

Acknowledgements

Figures 2, 3, 4 from HPGMG-FV by Sam Williams
Special thanks to Sam Williams, Kathy Yelick
Jim Demmel for his cs267 material on Multigrid