EXPERIMENTAL INVESTIGATION OF LIMITED-MEMORY SEGMENTAL REFINEMENT MULTIGRID

MATTHEW G. KNEPLEY*

Abstract.

1. Introduction.

Full multigrid (FMG) is a provably asymptotically exact, non-iterative algebraic equation solver for discretized elliptic partial differential equations with work complexity of about five residual calculations, or what is known as textbook multigrid efficiency (TME), for the constant coefficient Laplacian [?]. For some classes of elliptic problems, TME can be shown analytically, but it has been observed experimentally for a very broad spectrum of problems [?, ?, ?]. In addition, the Full Approximation Scheme (FAS) multigrid is applicable to general nonlinear elliptic equations.

Multigrid methods are widely used in practice, but distributed memory parallel multigrid presents many implementation challenges. The benefits for future computer architectures of the reduction in communication, measured in both power and time, are well-documented in [?]. Segmental refinement explicitly decouples subdomain processing on the finest multigrid levels, which improves data locality, amortizes latency costs, and reduces data dependencies. However, large surveys of the scientific computing userbase, such as [?], show that a majority of practitioners are confined to a desktop computer. In addition, those users with parallel code have very long running jobs ($> 50\%$ run for more than 24 hours), and make very little attempt to optimize the code, leading to the conclusion that the leading reason for parallelism is to run jobs too large for a single machine. Our limited memory implementation of SRMG can allow practitioners to run very large potential problems (10^{12} unknowns) in the same 24 hour timeframe on a standard desktop machine. This capability has the potential to revolutionize scientific computing for the vast mass of users.

Possible applications

2. Background.

Beginning in the 1970s, Brandt [?,?,?] proposed segmental refinement multigrid (SRMG) as a polylogarithmic memory alternative to traditional FMG that does not store the entire solution in memory at any one time. More recent work [?,?,?,?] has concentrated on the benefits for distributed memory computing, namely that SRMG has no interprocess communication on the finest grids. This paper presents an implementation with polylogarithmic memory requirements, and experimentally examines the accuracy, memory scalability, and dependence on interpolation order. We present multilevel numerical results, complementing [?], and verify complexity models for both computation and memory.

In prior work, it was unclear whether an actual implementation could scale indefinitely with a finite buffer size. In fact, contrary to the theory sketched in [?], in [?] it appears that the buffer size must grow with the number of levels. Second, no prior implementation had demonstrated polylogarithmic memory usage. We address both of these questions in Section ??.

3. Methodology.

We follow [?] in employing a vertex-centered discretization for the multigrid iteration, in this case a 5-point finite difference stencil on a Cartesian grid, embodied in the long-standing PETSc example code, SNES ex5 [?]. This code, in fact, solves the Bratu equation, but we reduce this to the Laplace equation by setting the parameter $\lambda = 0$.

*knepley@rice.edu, Department of Computational and Applied Mathematics, Rice University, Houston TX 77005

3.1. Low Memory Functionals. Since our implementation is low-memory by design, we must make use of the fine grid solution one patch at a time. Fortunately, we can do this by computing functionals $\mathcal{F}$ of the solution which can be expressed as integrals over the domain,

$$\mathcal{F}(u) = \int_\Omega f(u)\,dx. \tag{3.1}$$

Since the integral is additive, we can express this as a sum over patches

$$\mathcal{F}(u) = \sum_P \int_{\Omega_P} f(u)\,dx, \tag{3.2}$$

and we can batch this processing by computing several functionals at once on each patch. For example, we use the error functional $\mathcal{E}^p$ to check that our manufactured solution converges at the correct rate,

$$\mathcal{E}^p(u) = \left(\sum_P \int_{\Omega_P} |u - u^*|^p\,dx \right)^{1/p}, \tag{3.3}$$

so we integrate $|u - u^*|$, $|u - u^*|^2$, and $\max |u - u^*|$ simultaneously on each patch, keeping the result in a separate accumulator, and then after all patches at a given level are processed, we take the appropriate root and output the result. Thus, we need two functions for each user functional, one that computes the integral over a patch, and one that does final processing and outputs the result.

3.2. Higher Order Interpolation. Rujeko

3.3. Computational and Memory Models. Jeremy

3.4. Implementation. Frankie: documentation
Derek: build/test

4. Results. List all plots and the reason for each plot.
Michael: Mesh Convergence MMS 1, 2, 3, 4 (Correctness)
Ryan: Patch Convergence MMS 2, 3, 4 (Accuracy)
Jeremy: Memory Usage (Efficiency)
Ryan: Patch Convergence with different buffer size (Accuracy)
John: Patch Convergence with different interpolation order (Accuracy)
Shoeb: Runtime vs N (Efficiency)

5. Conclusions. Eric: Restate main result
future work, open problems

Acknowledgments. The authors acknowledge the generous support of the Department of Computational and Applied Mathematics at Rice Unviersity.