PERFORMANCE PORTABLE HPCG

ZACHARY A. BOOKEY*, IRINA P. DEMESHKO†, AND SIVASANKARAN RAJAMANICKAM‡

Abstract. The High Performance Conjugate Gradients (HPCG) Benchmark is an international project to create a more appropriate benchmark test for the world's largest computers. The current LINPACK benchmark, which is the standard for measuring the performance of the top 500 fastest computers in the world, is moving computers in a direction that is no longer beneficial to many important parallel applications. In this project we are developing a version of HPCG, using the Kokkos package found in Trilinos, that can be optimally executed across several distinct high performance computing architectures. This new code demonstrates an efficient programming approach that can be adopted by other programmers to write portable high performance software.

1. Introduction. The High Performance Conjugate Gradient, from here out referred to as HPCG, is an upcoming benchmark test to rank the worlds largest computers. HPCG uses a preconditioned conjugate gradient to solve a system of equations. The goal of our project was to create a version of HPCG that uses the Kokkos package found in Trilinos in order to increase portability among many types of architectures while maintaining reasonable performance.

2. HPCG. After generations of using the High Performance Linpack (HPL) benchmark to measure the performance of large computers it became necessary to use another benchmark to help better the direction that super computers were headed to more accurately reflect the types of applications that these machines were running. HPCG was created to fill the gap that HPL had created. On top of solving a large system of equations, HPCG also features a more irregular data access pattern so that data access affects results as well as matrix computations.

HPCG begins by creating a symmetric positive definite matrix and it's corresponding multi grid to be used in the preconditioning phase. For the preconditioner it uses a Symmetric Gauss-Seidel forward sweep and back sweep to solve the lower and upper triangular matrices. For the actual solve of $Ax = b$, HPCG uses the conjugate gradient method after the preconditioning phase.

HPCG runs in seven major phases.

1. **Problem Setup:** This is the beginning of HPCG and is where we construct the geometry that is used to generate the problem. HPCG generates a symmetric, positive definite, sparse matrix with up to 27 nonzero entries per row depending on the local location of the row.
2. **Validation Testing:** This portion of the program is to make sure any changes made produce valid results. Specifically it checks to make sure that both the unpreconditioned and preconditioned conjugate gradient converge in around 12 and 2 iterations respectively. It also makes sure that after performing both a sparse matrix vector multiplication and a symmetric Gauss-Seidel sweep that we preserve symmetry by using two pseudorandomly filled vectors and performing simple operations that should be zero due to the nature of our symmetric matrix A.
3. **Reference Sparse Matrix Vector Multiplication and Multigrid Timing:** This portion of the code times how long it takes to perform the reference

*Saint John's University, zabookey@csbsju.edu
†Sandia National Laboratories, ipdemes@sandia.gov
‡Sandia National Laboratories, srajama@sandia.gov

versionts of SPMV and Symmetric Gauss-Seidel.

4. **Reference Conjugate Gradient Timing:** Here we run 50 iterations of the reference version of the conjugate gradient method and record the resulting residual. This residual must be attained by the optimized version of conjugate gradient no matter how many iterations are required.
5. **Optimized Conjugate Gradient Setup:** Runs one set of the optimized conjugate gradient and determines the number of iterations required to reach the residual found before. Then figures out how mamy times to reach the desired residual to fill in the requested benchmark time.
6. **Optimized Conjugate Gradient Timing:** Runs the optimized conjugate gradient the required amount of times. Records time for each timed section to report out later.
7. **Report Results:** Writes out log files for debugging and creates the .yaml file to display the results which can then be submitted if all the requirements are met.

HPCG gives you the option to run with MPI, OpenMP, both, or in serial. Running with MPI adds an extra dimension to the problem and requires processes to exchange values on their borders to perform. This results in a tradeoff between more overhead and more parallelism.

3. Kokkos. As different computer architectures are better with certain applications than others it has become increasingly difficult to write code that will perform well across many different types of architectures. One solution to this problem is the C++ package, Kokkos. Kokkos acts as a wrapper around your code to allow you to specify at compile time where and how you want to run your application. Currently Kokkos supports the following execution spaces:

- Serial
- PThreads
- OpenMP
- Cuda

Kokkos has two main features, views and parallel kernels. A view is essentially a wrapper around an array of data that gives you the option to specify which execution space you want to store the data on and allows you to choose what sort of memory access traits you wish this data to have. Views also handle their own memory management via reference counting so that the view automatically deallocates itself when all of the variables that reference it go out of scope, thus making memory management much simpler across multiple devices.

There are three main parallel kernels: parallel_for, parallel_reduce, and parallel_scan. All of these serve their own purpose and act as wrappers over how you would execute a section of code in parallel over the respective exectuion space. For all of the parallel kernels you initiate the kernel by passing in a functor that performs the desired parallel operation, as of C++11 lambdas work as well and soon lamda functionality will be fully supported by cuda from host to device.

Parallel_for is simply a generic for loop that will run all of the context of the loop in parallel. This works well for parallel kernels like vector addition. Parallel_reduce is for simultaneously updating a single value, this function guarantees that you avoid race conditions with the updated values. Parallel_reduce works well for parallel kernels like finding the dot product of two vectors. Parallel_scan is for taking a view and creating a running sum of values to replace the values of the view. Although parallel_scan is useful it was only really needed for setup phases in our HPCG.

Kokkos allows for nested parallelism that involves creating a league of teams of threads. With this tool, a developer could launch a parallel_reduce kernel that uses a parallel_for to update some value that later gets added to the value that was initially included to be reduced. Although this has not been implemented in our project yet, there are places in our code that would benefit from nested parallelism and thus we intend to include it at a later time.

4. HPCG + Kokkos. The goal for our project was to create a version of HPCG that used Kokkos features to be able to produce valid results across many architectures.

5. Future Work.

6. Conclusion.

7. Actual Content. As we saw in Section 1, we all like chocolate pudding. This is where I wish \jargonfill worked. It would fill the page with meaningless technobabble so I could illustrate this package. Instead, I'll talk about how to use quotations in latex. "Never use these quotations." "Always use these, instead."

8. Conclusions. Herein, we repeat the abstract in past tense.

Unlike many other baked goods, chocolate pudding is subject to a myriad of interesting (and unique) effects on both the meso and nano scales. Understanding these phenomena is critical, not only to America's restaurant industry, but to children everywhere. We have examined these effects and have proposed new potential models which accurately capture the material structure of chocolate pudding.

REFERENCES

[1] M. A. Heroux, J. Dongarra, and P. Luszczek, *Hpcg technical specification*, tech. rep., Sandia National Laboratories.

[2] M. Mentor, *On chocolate pudding consumption rates*, Int. J. Quantum Foodstuffs, 42 (2006), pp. 42–142.