Towards Improving the Performance of the ADCIRC Storm Surge Modeling Software

Nick Weidner & Jijun Tang
Department of Computer Science & Engineering
University of South Carolina
Columbia, SC, USA
weidnern@email.sc.edu

Timothy Stitt
Center for Research Computing
University of Notre Dame
South Bend, IN, USA

Abstract-Accurately predicting storms and hurricanes is critical to saving lives and reducing economic loss. Therefore, it is necessary to use the most efficient software and hardware technology available in order to improve the performance and fidelity of these predictive mathematical models. For over ten years, the Computational Hydraulics Lab (CHL) at the University of Notre Dame has been involved in developing the high-resolution ADvanced CIRCulation (ADCIRC) storm surge model to predict storm surges in coastal areas. The objective of this work was to port a novel adaption of the parallel ADCIRC code to the state-of-the-art Intel Xeon Phi Co-Processor system (Stampede) at the Texas Advanced Computing Center (TACC) and ideally demonstrate speedup on a set of benchmark calculations. The porting process was accomplished by identifying fine-grained parallelism and vectorizable compute-intensive loops that could be offloaded to the 61-core Xeon Phi co-processors while leveraging their 512-bit wide vector units. Due to transfer latencies, offloading excessive amounts of code can reduce the effectiveness of using the Xeon Phi co-processors so the code was initially profiled in order to identify the code hotspots where the host processors spent the majority of their time (and demonstrated good potential for data parallelism). This analysis allowed us to focus the work of OpenMP and Xeon-Phi offloading on the most expensive routines. These routines were modified to take advantage of the full 16 threads available on the host processor, and will ultimately allow us to offload work to the increased number or threads on the partnered Xeon-Phis.

Keywords- Xeon Phi, Parallelism, Offloading

I. Introduction

Hurricanes are disastrous to coastal regions all across the world. They are especially dangerous if the storm is large, and the amount of preparation for the storm is small. Because of this, it is important to predict the actions of these storms before they happen, and evaluate the damages done by previous storms in order to prepare for ones still to come. The generalize goal of storm-surge modeling simulations is to better prepare individuals, governments and disaster-relieve agencies for storms in order to save lives and reduce property damages [1-5]. For example, hurricane Sandy, which measured 820 miles in diameter, hit New Jersey in October of 2012 and caused an estimated $25 billion in lost business activity, and left 8.1 million homes without power [6]. It is important for residents, agencies and municipalities to ready themselves for the storm to minimize the damages and negative impacts on coastal regions. Performing the appropriate calculations involves a large amount of computational power and time. For models to be effective and return timely, accurate results, they need to take advantage of the parallel computing power of super computers.

The high-resolution, ADvanced CIRCulation (ADCIRC) storm-surge model collaboratively developed at the Computational Hydraulics Lab (CHL) at the University of Notre Dame is an excellent example of highly parallel-processing code that is run on high-performance computers [4-5]. The ADCIRC model has been and continues to be the standard coastal model utilized by US Army Corps of Engineers (USACE) and Federal Emergency Management Agency (FEMA) [7]. It has been used, for example, to help reconstruct the levees in New Orleans after Katrina to prevent a future disaster of that magnitude [7,8]. Recent improvements to the ADCIRC code (reffered to here as DGSWEM) has changed the underlying numerical approach for calculating the results. This new numerical approach should provide more accurate results and stable performance. However, this increased fidelity could come at the expense of computational speed if the new code does not take advantage of the parallelism inherent in this new numerical approach. The parallel DGCWEM code divides up the region being simulated into a grid of user defined cells. It uses a message-passing interface (MPI) to communicate between these cells where each cell is given a processor to do calculations for it. The simulation equations are done independently for each cell on each processor and the MPI communicates any necessary data to its neighboring cells when needed. This is why large scale parallel computing is so important in this kind of simulation.

The ultimate objective of the work reported here was to port the DGSWEM storm-surge code to the state-of-the-art Intel Xeon Phi Co-Processor system (Stampede) at the Texas Advanced Computing Center (TACC) with the goal of showing increased performance on a set of benchmark calculations. To accomplish the objective the code was initially profiled in order to identify code hotspots, and these hotspots we the target of OpenMP implementation. When the results of the OpenMP were satisfactory enough that they outweighed the overhead of the implementation, these regions

could then exploit the parallel characteristics of the Xeon-Phis.

II. Results and Discussion

The purpose of porting the DGSWEM code to a Xeon-Phi system it to achieve considerable increase in wall-clock speed, and this can only be achieved by exploiting the Xeon-Phi technology effectively. The Xeon-Phi excels at running vectorizd code so the more code that can be vectorized the better. However, since the size of the source code is so large, it was necessary to narrow down which code segments would benefit most from offloading, and which segments had small enough execution times to remain on the host processor. Profiling allowed us to identify which code segments will benefit the most from offloading. Profiling was done using the University of Oregon's Tuning and Analysis Utilities (TAU) [9], and served two primary purposes. First we calculated the time spent in each individual source routine, for a set of benchmark runs, in order to determine which routines contributed most to the overall runtime. Secondly we profiled test runs at different processor counts in order to ensure that these hotspots remained significant as we increased the processor count. If this was not the case, then we would want to reevaluate where to focus our offloading since most realistic simulations are performed at high processor counts (typically 2000 – 16,000 cores). All of these runs were performed solely on the host processor and only wall-clock times were calculated. These profiling results allowed us to determine where to focus our offloading efforts. Also, when we did achieve an offload result we could re-profile it and compare the runtimes between the original runs solely on the host and the new runs with particular routines offloaded to the Xeon-Phi.

With our routines chosen, we began implementing OpenMP and dividing work up amongst the host processors threads. This meant designating which data values in each routine were either shared or private to the individual thread they would be working on. This meant classifying each individual variable used in each routine. We mentioned earlier that dividing up this kind of work worked best on code that was vectorized. The problem is that our largest routine was not completely vectorized. Because this routine contains shared data values that are read/written from multiple threads, atomic statements were added to the code to prevent different threads from having the same value. The run time results of the code with OpenMP implemented over a number of threads show that adding the atomic statements add additional computation time. However, this overhead is overcome once we have at least 2 threads implemented, and by 16 threads DGSWEM runs 4 times faster than a single thread. This is still slower than the old ADCIRC with the less accurate numerical approach, but we can push the limitation of 16 host processor threads by taking advantage of the additional ones on the partnered Xeon-Phis. Therefor the next step is implementing Xeon-Phi offloading to take advantage of the additional threads.

Identify applicable sponsor/s here. If no sponsors, delete this text box

References

1. Westerink, J. J., R. A. Luettich, A. M. Baptista, N. W. Scheffner, P. Farrar, "Tide and Storm-Surge Predictions Using Finite-Element Model," *J. Hydraulic Engineering*, **118(10)**, 1373-1390 (1992).
2. Bode, L., and T. A. Hardy, "Progress and Recent Development in Storm Surge Modeling," *J. Hydraulic Engineering*, **123(4)**, 315-331 (1994).
3. Hubbert, G. D., and K. L. McInnes, "A Storm Surge Inundation Model for Coastal Planning and Impact Studies," *J. Coastal Research,* **15(1)**, 168-185 (1999).
4. Dawson, C., E. J. Kubatko, J. J. Westerink, C. Tranhan, C. Mirabito, C. Michoski, N. Panda, "Discontinuous Galerkin Methods for Modeling Hurricane Storm Surge," *Advances in Water Resources*, DOI 10.1016/j.advwatres.2010.11.004, **34**, 1165-1176 (2011).
5. Tanaka, S., S. Bunya, J. J. Westerink, C. Dawson, R. A. Luettich, "Scalability of an Unstructured Grid Continous Galerkin Based Hurricane Storm Surge Model," *J. Scientific Computing*, **46**, 329-358 (2011).
6. Webley, K., "Hurricane Sandy By the Number: A Superstorm's Statistics, One Month Later," (November 26, 2012), Retrieved from: http://nation.time.com/2012/11/26/hurricane-sandy-one-month-later/.
7. "Flood Insurance Study: Southeastern Parishes, Louisiana, Intermediate Submission 2: Offshore Water Levels and Waves," FEMA, US Army Corps of Engineers, New Orleans District, July 24, 2008.
8. Link, L. E., J. J. Jaeger, J. Stevenson, W. Stroupe, R. L. Mosher, D. Martin, J. K. Garster, D. B. Zilkoski, B. A. Ebersole, J. J. Westerink, D. T. Resio, R.G. Dean, M. K. Sharp, R.S. Steedman, J. M. Duncan, B. L. Moentenich, B. Howard, J. Harris, S. Fitzgerald, D. Moser, P. Canning, J. Foster, B. Muller, "Performance Evaluation of the New Orleans and Southeast Louisiana Hurricane Protection System," Volume I – Executive Summary and Overview, Draft Final Report of the Interagency Performance Task Force, U.S. Army Corps of Engineers, Washington, D.C., June 2008.
9. Shende S., and A. D. Malony, "The TAU Parallel Performance System," *Int. J. High Perform. C.*, **20(2)**, 287-331 (2006).