Solution of the Time-Dependent Acoustic-Elastic Wave Equation on a Heterogeneous, Coprocessor-Enabled Supercomputer

Jesse Kelly Institute for Computational Engineering and Sciences
The Univeristy of Texas at Austin
Austin, TX, USA
jkelly@ices.utexas.edu

Hari Sundar Institute for Computational Engineering and Sciences
The Univeristy of Texas at Austin
Austin, TX, USA
hari@ices.utexas.edu

Omar Ghattas Institute for Computational Engineering and Sciences
The Univeristy of Texas at Austin
Austin, TX, USA
omar@ices.utexas.edu

ABSTRACT

Modern supercomputers increasingly utilize accelerators and coprocessors. A key challenge in using such heterogeneous systems has been achieving load balance such that neither the CPU nor the coprocessor is left idle. Traditional approaches have offloaded entire computations to the coprocessor, resulting in an idle CPU, or have opted for task-level parallelism requiring large data transfers between the CPU and the coprocessor. True work-parallelism has been hard as coprocessors cannot directly communicate with other CPUs (besides the host) or other coprocessors. We present a new nested partition scheme to overcome this problem. By partitioning the work assignment on a given node asymmetrically into boundary and interior work, and assigning the interior to the coprocessor, we are able to achieve excellent efficiency while ensuring proper utilization of both the CPU and the coprocessor. The problem used for evaluating such partitioning is an hp-discontinuous Galerkin spectral element method for coupled acoustic-elastic wave propagation.

Keywords

load balancing, instruction-level and thread-level parallelism, special-purpose architectures, coprocessor, heterogeneous, MIC, hybrid

1. INTRODUCTION

As hardware manufacturers bump against the upper limits of processor speed, the way forward in gaining more performance seems to be to move toward multi- and many-core systems. Increasingly, the supercomputing community is seeing large systems that realize the bulk of their computational power from coprocessors [2]. According to the June 2013 Top500 list, four of the top ten highest-performing supercomputers in the world made use of coprocessors to achieve a significant amount of their computational power [1]. In 2012 Intel® presented their Xeon Phi™ family of coprocessors, based on the Many Integrated Core (MIC) architecture. The MIC lives on the PCI bus, and like a GPU it communicates with the host CPU through special directives and maintains its own GDDR memory. Unlike a GPU, the MIC is composed of many in-order x86 cores with 512-bit wide vector registers and floating-point units. The x86 architecture is a widespread, familiar architecture that enjoys support by a large number of compilers, libraries, and programming expertise. The MIC thus allows programmers to use familiar tools such as C, FORTRAN, OpenMP and MPI to implement parallelism. The work presented in this poster makes use of the Stampede supercomputer housed at the Texas Advanced Computing Center. Stampede is a 10 PFLOPS system which derives 8 PFLOPS from the Xeon Phi™; it consists of 6400 compute nodes, each of which houses two 8-core Intel® Xeon™ E5 CPUs and up to two 61-core Intel® Xeon Phi™ SE10P coprocessors.

Table 1: Key performance characteristics of Stampede compute node.

	MIC	2x 8-Core Xeon E5
Peak DP GFLOPS	1070	346
Peak Bandwidth (GB/s)	320	102.4
Memory (GB)	8 GDDR5	32 DDR3

The floating-point operation (FLOP) rate and theoretical peak memory bandwidth on the MIC outperforms the CPU by approximately a factor of three (See Table 1). As with most coprocessors, the amount of work that can be done by the MIC is constrained by its limited amount of random-access memory. To begin with, the MIC has only 25% as much RAM as is available to its host-node CPUs. Considering the MIC's higher core count the situation becomes worse, as there are approximately 134 MB of RAM for every MIC core, compared to 2 GB of RAM for each CPU core. Thus, the MIC is most effectively used as a large SMP node, programmed using a shared-memory paradigm. Starting with a distributed memory finite-element code implemented in pure MPI, we optimize the solution of the acoustic-elastic wave equation for execution on both the Xeon™ and the Xeon Phi™. We achieve significant speedups relative to

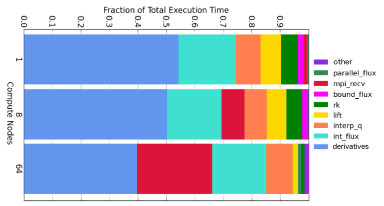

Figure 1: Fractions of overall runtime spent in each kernel for the baseline, pure-MPI code.

Table 2: Speedup of optimized, accelerated strong scaling runs relative to baseline MPI.

Cores	MICs	7th Order	15th Order
32	4	5.78x	6.88x
64	8	5.46x	4.51x
128	16	4.08x	4.15x
256	32	3.44x	3.44x
512	64	2.75x	2.75x
1024	128	2.33x	1.89x

the pure MPI code through the use of SIMD vector instructions, OpenMP, and offloading to the MIC. This extended abstract will briefly present the problem being solved, our optimization and partitioning scheme for MIC offload, and performance results.

2. PROBLEM

We use a discontinous Galerkin finite element method [3] to solve the first-order system of equations:

$$\rho\frac{\partial \mathbf{v}}{\partial t} = \nabla\cdot(\lambda tr(\boldsymbol{E})\boldsymbol{I} + 2\mu\boldsymbol{E}) + \mathbf{f} \tag{1a}$$

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{2}(\nabla\mathbf{v} + \nabla\mathbf{v}^T) \tag{1b}$$

which describes the propagation of seismic waves through isotropic elastic media. The acoustic wave equation can be recovered by setting μ to zero. The numerical solution of this system using a discontinuous Galerkin method is composed of several key kernels. The execution times for each of these kernels is shown in Figure 1.

3. OPTIMIZATION & PARTITIONING

We first optimized the execution of the pure-MPI code using SIMD vector instructions. This is especially important on the MIC, as each core has a VPU that is capable of 16 double-precision floating point operations per clock cycle (for a fused multiply-add). Many kernels were auto-vectorizable, and as such vectorization was as simple as ensuring proper data alignment and adding compiler directives to the appropriate loops. The "derivatives" kernel, however, which makes up the bulk of the time spent in the execution of each time step, was selected for hand-vectorization.

We next restructured our solver as a hybrid MPI-OpenMP code in anticipation of MIC offload. Spawning one MPI process per compute node makes load balancing and MIC control much easier, as each compute node has only one MIC. A one-to-one pairing between a MIC and an MPI process eliminates the additional communication and synchronization that would be required were multiple MPI processes to try offloading to the same MIC. This strategy also allows us to run the same core solver code on both the CPU and the MIC, as both will now use OpenMP to make use of the multicore hardware on the CPU or MIC. The only adjustment necessary to the code is in the communication stage of the solution process; on the MIC, MPI calls are replaced with compiler directives for offload communication. Each MPI process handles assigning a subset of its elements to exactly one MIC, spawns OpenMP threads to make use of all CPU cores, and mainains communication with both the MIC and neighboring MPI processes.

This scheme requires that the elemental subdomain owned by the MIC only shares faces with its host MPI process. The partitioning algorithm used by the baseline MPI code does not allow for control over partition geometry or connectivity to other partitions, and so a second-level partitioning scheme must be implemented. Maintaining the uniformly-balanced partitioning scheme at the MPI level, we then run our second partitioning algorithm on each compute node to construct a subdomain of elements to be offloaded to the MIC. The presentation of our second-level partitioning algorithm exceeds the scope of this abstract, but please see our poster for details.

4. RESULTS

We observed weak scaling as good as or better than our baseline, pure-MPI code. Strong scaling with coprocessor-accelerated software is difficult, because as the number of compute nodes increases, the ratio of exterior faces to elements increases on each computational subdomain. This leaves fewer "interior" elements that are eligible to be offloaded to the MIC, and thus we see the performance of the accelerated code approach the performance of the non-accelerated code in the limit as the number of processors increases for a fixed problem size.

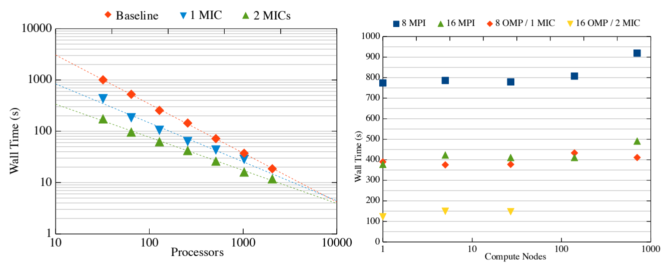

Figure 2: Strong/weak scaling, 7th order elements.

5. REFERENCES

[1] Top500 website, June 2013.

[2] V. Kindratenko and P. Trancoso. Trends in high-performance computing. *Computing in Science Engineering*, 13(3):92–95, 2011.

[3] L. Wilcox, G. Stadler, C. Burstedde, and O. Ghattas. A high-order discontinuous galerkin method for wave propagation through coupled elastic–acoustic media. *Journal of Computational Physics*, 229(24):9373–9396, 2010.