Sustainable Computing: Informatics and Systems 10 (2016) 36–47

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

A genetic algorithm based approach for multi-objective hardware/software co-optimization

Tania Banerjee*, Mohamed Gadou, Sanjay Ranka

Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, United States

A R T I C L E   I N F O

Article history:
Received 13 December 2015
Received in revised form 4 April 2016
Accepted 5 April 2016

Keywords:
Spectral element methods
Power and energy evaluation
Performance benchmarking
Exascale
Hardware/software co-optimization

A B S T R A C T

We develop a genetic algorithm based autotuning strategy in this paper. Autotuning is a platform independent code optimization process in which different hardware and software parameters of the code being optimized are identified and the parameter space explored to arrive at an alternative implementation that optimizes characteristics such as performance and energy consumption. The main advantage of our approach is that the number of possible compilations and executions that are explored in the configuration space is substantially smaller than exhaustive search. We demonstrate the usefulness of our approach to the underlying small matrix multiplication routines in spectral element solvers. The latter are an important class of higher order methods that are expected to be computationally intensive portion of the next generation of large scale CFD simulations. Our experimental results were conducted on a variety of existing platforms as well as on gem5 simulator platform with different cache configurations. On an existing platform, AMD Fusion, the genetic algorithm is able obtain 34% improvement in performance and 37% reduction in energy consumption over existing versions of the code. The fact that a very small fraction of the entire configuration space needs to be explored becomes very useful as algorithmic exploration is combined with exploration of cache configuration resulting in hardware/software co-optimization. We used the micro-architectural simulator, gem5, to evaluate different cache configurations for energy and performance trade-offs for out-of-order x86 cores at the micro-architectural level for small matrix multiplications. Our results show how genetic algorithm based autotuning strategy can come up with a close to optimal variant analyzing only about 0.25% of the exploration space.

1. Introduction

The increasing complexity of computer architecture has made it very difficult to program codes with optimum performance. Processor features such as vector units, multithreading and pipelining requires varied and complex optimization techniques that are rather difficult to handle either automatically by today's compilers or manually by the programmer. The problem further aggravates when we have multiple and often conflicting objectives for optimization. A practical method of optimizing these codes is autotuning. Autotuning is a method of empirical optimization in which the configuration space of a code is automatically explored in a systematic manner along with hardware parameters that impact performance and energy requirements of the algorithm, thereby finding the best values for these parameters on a given platform. Autotuning results in highly tuned codes for specific architectures and has the potential for significantly improving performance and energy requirements. Attractive though it is as an optimization technique, autotuning involves exhaustive exploration of parameter space which is time consuming. As the number of tunable parameters is increased, the exploration space increases exponentially and exhaustive search quickly becomes infeasible.

We developed a genetic algorithm (GA) based search mechanism that efficiently explores the parameter space to obtain close to optimal parameter settings for a given code. GA is a technique for solving optimization problems with or without constraints and imbibes the principles of natural selection followed in biological evolution [1]. GA has been used in a vast area of applications including multiple sequence alignment in bioinformatics [21], to find out weak links in approximate computing in computer architecture [22], and in natural language processing [23]. We use our optimization strategy on spectral element method (SEM) which is a high-order weighted residual technique that provides the geometric flexibility of finite elements, rapid convergence properties and tensor-product efficiency benefits of global structure methods [2]. They are expected to form the core computation of a large number of applications targeted for exascale computing. NEK5000, a

* Corresponding author.
E-mail addresses: tmishra@cise.ufl.edu (T. Banerjee), mgadou@cise.ufl.edu (M. Gadou), ranka@cise.ufl.edu (S. Ranka).

http://dx.doi.org/10.1016/j.suscom.2016.04.001
2210-5379/© 2016 Elsevier Inc. All rights reserved.

publicly available software, is based on SEMs and is expected to be used in a variety of applications for simulating fluid flow, heat transfer and magneto hydrodynamics as well as electromagnetics [7]. CMT-Nek, an ongoing development effort at the University of Florida for compressible multiphase turbulence, is based on NEK5000. One of the most compute intensive kernels of SEM is derivative computation involving small matrix matrix multiplications and CMT-nek is expected to compute derivatives for 25–50% of its runtime.

Matrix matrix multiplication has been studied extensively in literature for large matrices. SEM, on the other hand, uses small matrices (generally of sizes between 4×4 and 16×16). Achieving high performance matrix multiplications for small matrices is significantly more challenging than larger matrices as the number of times data elements are copied from memory/cache to registers is relatively small. Thus the computation to data access requirements is significantly higher for smaller matrices. Our goal in this paper is to obtain high performance and energy efficient implementations of SEM methods for a variety of existing architectures, as well as to explore hardware/software configuration space, thereby demonstrating the efficiency of our autotuning method. In particular, we use loop unrolling and loop permutation for code transformation with the relevant software parameters being loop unroll factors and permutation sequences. We also use hardware parameters, cache size, line size and set associativity, for different cache configurations.

The main contributions of this paper are as follows:

1) We present a genetic algorithm based driver for our autotuning approach which effectively navigates through the large configuration space of the algorithms being optimized. This approach reduces the number of configurations that need to be compiled and experimentally evaluated as compared to an exhaustive approach. The approach can be used to optimize performance and/or energy and potentially a combination of these or other objectives.
2) Using the genetic algorithm based approach, we develop improved implementations of the NEK5000 SEM algorithm for multi-core architectures of traditional CPU cores for performance and energy. In particular our best codes result in implementations on AMD Fusion cores that are faster by 34% on an average as compared to the original manually tuned version of NEK5000. The amount of energy requirements are also about 37% lower on an average.

   Our results also show performance and energy requirements are highly correlated for the target architectures. Thus, multi-objective optimization based on performance and energy can be greatly simplified as optimizing for performance generally results in better energy requirements and vice versa.
3) We demonstrate an efficient exploration of hardware/software design space for SEM applications using GA based search. Inclusion of hardware parameters causes the search space to become more huge. Our experiments show that GA based search can come up with close to optimal variant after evaluating only 0.25% of the search space.

The rest of the paper is described as follows. In Section 2, we briefly describe the spectral methods in the context of NEK5000. Section 3 provides our genetic algorithm based autotuning and optimization method for the underlying small matrix multiplications on traditional CPU cores and on gem5 simulator framework. Section 4 provides experimental evaluation and we conclude in Section 5.

2. Spectral element methods and NEK5000

NEK5000 [2] is a scientific code for modeling incompressible flows using computational fluid dynamics solver based on SEM. The code was recognized for its algorithmic quality and sustained parallel performance in 1999 with the Gordon Bell prize. It has been shown that the code scales to over a million cores.

One of the most compute intensive section of the NEK5000 code is the spectral element solver which performs matrix–vector multiplication repeatedly for each spectral element. The tensor product structure of the SEMs allows the matrix–vector products to be computed as a sequence of matrix–matrix multiplications, which are used to numerically compute partial derivatives of a function U, represented as u_{ijk} in three dimensional space. The derivatives of U along the Cartesian coordinates (r, s, t) are given by the following equations,

$$\left.\frac{\partial U}{\partial r}\right|_{i,j,k} = \sum_{l=1}^{N_x} A_{il} u_{ljk} \tag{1}$$

$$\left.\frac{\partial U}{\partial s}\right|_{i,j,k} = \sum_{l=1}^{N_y} B_{jl} u_{ilk} \tag{2}$$

$$\left.\frac{\partial U}{\partial t}\right|_{i,j,k} = \sum_{l=1}^{N_z} C_{kl} u_{ijl} \tag{3}$$

where an element in three dimensional space is assumed to have $N_x \times N_y \times N_z$ Gauss–Lobatto quadrature points with values of U being defined at those points. A, B, C are the derivative operators, of dimensions $N_x \times N_x$, $N_y \times N_y$ and $N_z \times N_z$ respectively. If $N_x = N_y = N_z = N$ then, $B = C = A^T$. From Eqs. (1), (2) and (3) we observe that for computing these derivatives, the fastest changing index in U are the first, middle and last indices respectively. Thus the memory access pattern for matrix U is different for each of the derivatives. For example, for computing partial derivatives along r, contiguous memory addresses are accessed, whereas for computing those along s and t, memory is accessed in strides of N_x and $N_x \times N_y$ addresses respectively. N_x, N_y, N_z represent order of the polynomials that are fitted and values between 5 and 25 yield highly accurate results. Thus instead of large matrices, SEM requires the solution of a large number of matrix multiplications of very small matrices. We address efficient solutions for this problem in this paper.

The basic set of equivalent codes for computing partial derivatives along r are given in Fig. 1. We apply loop transformations to the algorithms of Fig. 1 and benchmark the resulting implementations for power and performance. For computing partial derivatives along the s and t directions, similar sets of codes are used, except, memory access patterns for the *4loop* version of partial derivatives along s, prevents loop fusion – so, a *4loop-fused* version of code does not exist for this derivative.

We impose the following restrictions on the configuration space based on NEK5000 specifics and compiler characteristics:

1) The matrix dimensions are considered identical in each direction. Thus, $N_x = N_y = N_z = N$ for a three dimensional matrix.
2) Unroll factors are divisors of the matrix sizes considered. This is helpful in eliminating any residual code that would be necessary otherwise. Processing the residual code for small matrix multiplications is a non-negligible overhead.
3) The unroll factors are limited by an amount that is likely to fill the L1 instruction cache. When the entire code fits the L1 instruction cache, performance is much better since otherwise instruction cache misses would impact performance.