Introduction

Load balancing irregular computations is a serious challenge in scientific simulation, especially in light of today’s supercomputers having over 1 million processor cores. It is becoming increasingly more difficult to exploit available parallelism without succumbing to processor starvation and the prohibitive overhead inherent to dynamic load balancing. This work presents a novel load balancing technique based on performance modeling and static partitioning as applied to NWChem’s coupled cluster (CC) module. Many chemical problems related to combustion, energy conversion, and molecular spectroscopy are untenable without CC methods on supercomputers. Given the steep computational requirements of CC, its scalability is crucial for supporting innovative science. The “Inspector/Executor” technique is shown to reduce the NWChem CC module’s execution time by as much as 50% at scale. While the technique is shown to reduce the NWChem CC module’s execution time by as much as 50% at scale, many chemical problems related to combustion, energy conversion, and molecular spectroscopy are still prohibitive if the computational requirements of CC are not met.

A TCE-CC simulation of a chemical system (such as a water molecule cluster) consists of a large collection of computational tasks, each containing at least one call to DGEMM and TCSORT. Fig. 3 shows that the number of floating point operations drastically varies from task to task. Load balance is achieved with a NXTVAL counter which dynamically assigns tasks to available processors.

Static Partitioning Based on Performance Models

In NWChem’s TCE-CC module, the cost of each task (its execution time) is estimated using performance models. The dominant computational routines: DGEMM and TCSORT. The DGEMM model:

\[
\frac{(m + n + k)^3}{3} + (n + k) + \frac{(m + n + k)d}{2}
\]

is the canonical approximation of matrix multiplication of A (having dimensions m by k) and B (dimensions k by n) where a, b, c, and d are system-dependent coefficients. The model’s terms correspond to the m*n d store products of length k, the m*n store operations, the m load operations of size k from A, and the n load operations of size k from B. The leading coefficients are found by solving a nonlinear least squares problem using standard methods.

The TCSORT routine arises in this application because external indices must be in ascending order across two tensors to be contracted (see [1] for details). The TCSORT performance model is based on empirical measurements conducted offline (Fig. 8). Using these empirical data, a performance model is constructed by performing a cubic fit of the data for all possible permutations.

Static partitioning is accomplished by performing the Longest Processing Time (LPT) algorithm [4]. The LPT has been proven to be a 4/3 approximation algorithm, meaning that it always produces a result which is 4/3 times the optimal solution. While our implementation uses LPT, any partitioning method can be applied when using the Inspector/Executor algorithm for load balancing.

Background

The NWChem toolkit supports a wide range of chemistry simulation methods with controllable accuracy on many of the latest supercomputer architectures. It is well-known for its quantum mechanical modules which are relatively scalable on large parallel systems.

The CC module allows for detailed study of chemical systems by iteratively solving the Schrödinger equation with an accurate ansatz. In particular, we are looking at the CC module produced by the Tensor Contraction Engine (TCE) [1]. The TCE automates the derivation and parallelization of the CC equations. In these TCE-CC codes, computations primarily consist of tensor permutations (TCSORT’s) and Basic Linear Algebra Subroutine (BLAS) matrix multiplications (DGEMM’s).

NWChem is built on Global Arrays (GA) (Fig. 1), which provides a shared memory style programming interface for distributed memory systems [2]. GA supports large scale development of one-sided messaging applications, but its standard template for dynamic load balancing (NXTVAL) is centralized in nature. NXTVAL, in essence, is a dynamic counter that controls task assignment (Fig. 2).

Inspector/Executor Motivation and Design

A TCE-CC simulation of a chemical system (such as a water molecule cluster) consists of a large collection of computational tasks, each containing at least one call to DGEMM and TCSORT. Fig. 3 shows that the number of floating point operations drastically varies from task to task. Load balance is achieved with a NXTVAL counter which dynamically assigns tasks to available processors.

Because NXTVAL is a central counter occupying a single memory location, its load balancing overhead can be substantial. This is shown in Fig. 4 where the TAU performance system was used to gather an inclusive-time profile of a 14 water molecule cluster simulation with approximately 900 MPI processes. The NXTVAL routine consumes 37% of the application. Fig. 5 shows the time per call to NXTVAL rises as the number of MPI processes increases.

However, using the Inspector/Executor algorithm for load balancing.

Fig. 9 shows the strong scaling performance of the original NWChem TCE-CC code versus the simple version of the Inspector/Executor for a highly accurate water molecule simulation. Fig. 10 is a similar but less accurate experiment for benzene. The benzene simulation also shows the benefit of using a “hybrid” approach which applies NXTVAL to contractions that are empirically shown to be faster than the Inspector/Executor.

For relatively dense problems such as a 14-molecule water cluster, the more advanced Dynamic Buckets (DB) technique is required to see performance improvements (Fig. 11). DB improves performance by as much as 15% in this case because it amortizes the effect of system noise on task execution time estimations by assigning collections of tasks to groups of processor cores.

Conclusions

The Inspector/Executor algorithm reduces the overhead from centralized dynamic load balancing in NWChem’s TCE-CC module. This benefit is due to the effects shown in Fig. 12: the overhead from NXTVAL (yellow) is reduced to the overhead of performing the cost estimation and static partitioning (red). After iterative refinement converges, subsequent iterations have no load balancing overhead (as in the final row of Fig. 12). Relatively worse load balance is traded for a drastic reduction in overhead.

The Dynamic Buckets approach further reduces execution time by improving load balance. For example, the load imbalance in the lower two rows of Fig. 12 could be improved by partitioning tasks across groups of processors, rather than individual processors. Because some task costs are over-estimated, and some under-estimated, better load balance is achieved when using Dynamic Buckets.

References


contact: ozog@cs.uoregon.edu