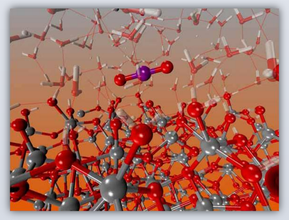
Inspector/Executor Load Balancing Algorithms for Block-Sparse Tensor Contractions

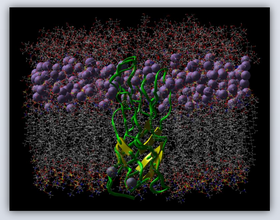
David Ozog[1], Jeff Hammond[2], Pavan Balaji[2],
James Dinan[2], Sameer Shende[1], Allen Malony[1]

University of Oregon[1] Argonne National Laboratory[2]

Introduction

Load balancing irregular computations is a serious challenge in scientific simulation, especially in light of modern 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 implementation is presented within the context of NWChem, this method is applicable to any application requiring load balance where reasonable estimations of computational kernel execution times are available.





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 (TCE_SORT'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).

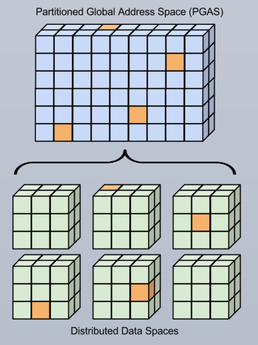


Fig. 1 : Global Arrays

```
Global Arrays: A, B
Local Buffers: a, b
count = 1
next = NXTVAL()
for i = 1 : N do
  if ( next == count ) then
    Get A(i) into a
    b = FOO(a)
    Update B(i) with b
    next = NXTVAL()
  end if
  count = count + 1
end for
```

Fig. 2 : NXTVAL Template

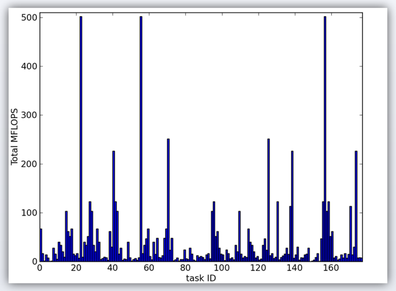


Fig. 3 : FLOP Imbalance

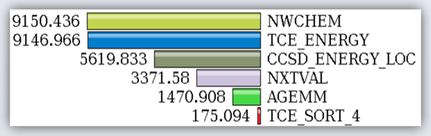


Fig. 4 : Inclusive-time in seconds

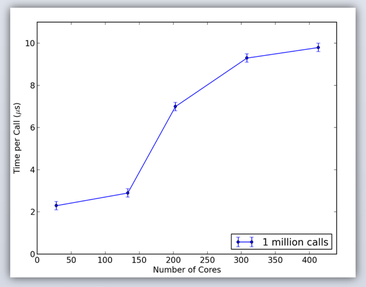


Fig. 5 : Time per Call to NXTVAL

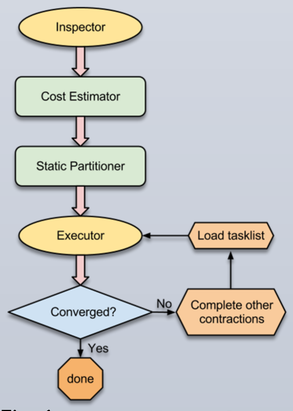


Fig. 6 : Inspector/Executor Design

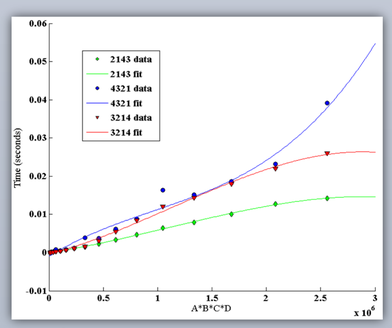


Fig. 8 : TCE_SORT performance measurements

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 TCE_SORT. 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.

We propose a low-overhead alternative to NXTVAL load balancing which uses task cost estimation and static partitioning. The design is presented in Fig. 6, where an inspector component performs a preliminary collation of tasks, then feeds them into a cost estimator (described in the next section). After partitioning the tasks into groups consisting of equal cost loads, an executor assigns and manages tasks. This straightforward approach is improved with the "Dynamic Buckets" design, shown in Fig. 7. Here, tasks are split into buckets which are then associated with groups of processors. This is done in an effort to minimize the detrimental effect of variation in task execution times.

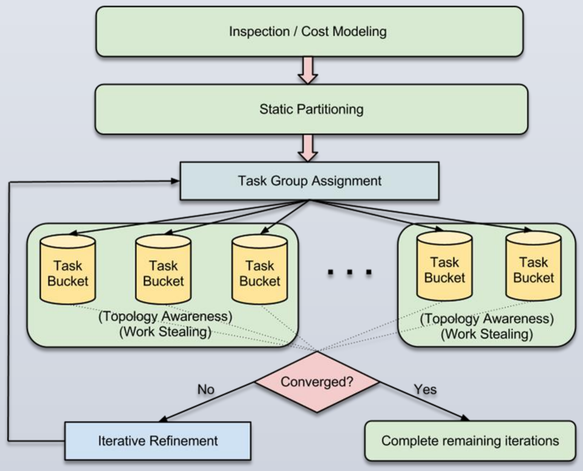


Fig. 7 : Dynamic Buckets Design

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 of the dominant computational routines: DGEMM and TCE_SORT. The DGEMM model:

$$t(m,n,k) = a(mnk) + b(mn) + c(mk) + d(nk)$$

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* dot 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 [3].

The TCE_SORT routine arises in this application because external indices must be in ascending order across two tensors to be contracted (see [1] for details). The TCE_SORT 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 each possible permutation:

$$p(x) = p_1(x)x^3 + p_2(x)x^2 + p_3x + p_4$$

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.

Results

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 nitrogen 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.

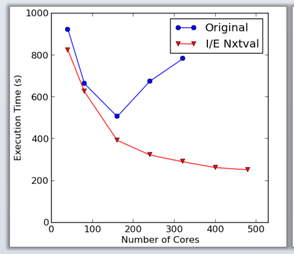


Fig. 9 : Nitrogen molecule (T)

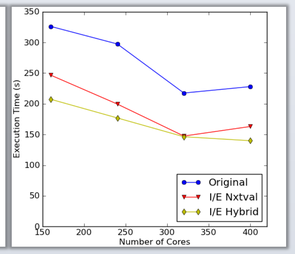


Fig. 10 : Benzene Monomer (D)

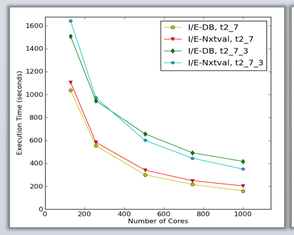


Fig. 11 : 14-H_2O - Dynamic Buckets

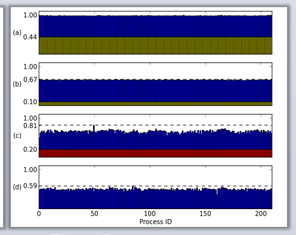


Fig. 12 : Load Balance and Overhead

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

[1] So Hirata. Journal of Physical Chemistry A, 107:9887-9897, 2003.
[2] J. Nieplocha, R. Harrison, et. al. SC 94, pg. 340-349, 1994.
[3] D. Marquardt. Journal of the Society for Industrial and Applied Mathematics, 11(2):431-441, 1963.
[4] R. L. Graham. SIAM Journal on Applied Mathematics, 17(2):416-429, 1969.

contact: ozog@cs.uoregon.edu

Introduction photos from:
http://www.nwchem-sw.org

This poster has been created in part by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DEAC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.