A Load Balancing Strategy for Computations on Large, Fixed Data Sets

Jan Christian Meyer

Norwegian University of Science and Technology,
Dept. of Computer and Information Science,
Sem Sælands v.7-9, NO-7034 Trondheim, Norway
Jan.Christian.Meyer@idi.ntnu.no

Abstract. Balancing computational loads is a topic where the conflicting demands of general and optimal solutions result in a great variety of strategies, each tailored to the specific needs of the application in question. This work develops a method for balancing a computational load where large sets of input data may be examined in advance, but where requests on this set may appear and change continuously, affecting the system load. The approach is based on approximating the computational cost by a polynomial, and estimating the load as an integral of this expression.

1 Background

The work originates from the need to adapt an existing load balancing scheme for a server which animates visualisations of geological data sets in real-time to suit a multi-user environment. The original system relies on a static subdivision of its input data in order to distribute the computational load on a network of dedicated computers. The nature of this approach makes it ill-suited for multi-user requirements, as it effectively assumes that the computational tasks from each user are best distributed across the entire array of machines, and does not consider variations in the data set sizes and computational complexity of the requests.

The developed approach follows the pattern of a generic description of dynamic load balancing strategies due to Piersall and Elfayoumi [1]. It is, however, not a pure dynamic solution; it also borrows a feature from semi-static load balancing schemes to exploit the application specific property of read-only access to the data set.

Load balancing methods may be classified in terms of 3 broad categories, with the following characteristic properties:

- **Static** load balancing relies on a predetermined optimal subdivision of the computational load, which can be embedded in the program task to be balanced.

- **Semi-static** load balancing relies on some measured or determined system state which is known at the beginning of run-time, and which allows the program to adjust it's parameters to balance the load before execution.
- **Dynamic** load balancing continually reconfigures the distribution of the computational load while the program is running.

The process of dynamic load balancing breaks down into the following phases:

1. **Load evaluation** involves estimating whether the present state of the computational load warrants proceeding with an effort to bring it into balance.
2. **Profitability determination** involves evaluating the cost of maintaining an imbalanced load, and comparing it to the cost involved in bringing it into balance.
3. **Work transfer calculation** involves determining the work transfer set, i.e. the set of tasks which can be transferred or exchanged in order to bring the computation into balance.
4. **Task selection** involves selecting a set of tasks and transfers which will satisfy the work transfer set generated in the previous phase.
5. **Task migration** consists of carrying out the work transfer decided upon in the previous phase.

2 Model

The work introduces a simplified theoretical model of a distributed-memory parallel computer, a data set, and a program to run on this set. The generic model is adapted to the special case of homogenous and fully interconnected computational clusters, as this is the class of systems the visualisation server is intended to run on.

The system as a whole is modelled in terms of

- a set of machines
- a program, expressed as a set of functions
- a set of data
- a cost function which maps the application of a function to a data subset to the time it requires to run
- a transfer constant which expresses the time required to move a data subset from one machine to another

The work proposes computationally efficient strategies for the 5 phases of dynamic load balancing, expressed in terms of this model. The consideration is made that computational efficiency in the load balancing algorithm is more important than the optimality of its results in a real-time context. In particular, it may be noted that Watts [2] points out the general task selection problem as NP-complete. Since this constitutes one of the phases of dynamic load balancing, a general and optimal algorithm is likely to be computationally intractable.

The main contribution of this work is to exploit the read-only nature of the data sets to be visualised, in order to prepare an appropriate load function which can be efficiently computed. It is argued that the cost function for fixed sets of functions and data may be approximated by polynomial functions in practically all cases, and that a computational load consisting of a combination of functions and data from these sets may be modelled by evaluating definite integrals resulting from the (analytic) integral of this approximate cost function. Solving a general definite integral with respect to its bounds provides a technique for partitioning the computational load into equally expensive tasks.

Specifically, let $\lambda f(X)$ denote the application of function f to the domain $X = x_1, ..., x_n$, and let $c(\lambda f(X))$ be the polynomial approximating the cost of this evaluation. The computational load of evaluating f across all of X may then be defined as

$$l_c = \int c(\lambda f(X)) \tag{1}$$

or, for some bounded subproblem,

$$l_c(a, b) = \int_a^b c(\lambda f(X)) \tag{2}$$

As c is a polynomial, it is easily integrated analytically, yielding an expression for $l_c(a, b)$ which may be solved with respect to b. By initialising a as the lower bound of X, this lets us specify a desired load value to find a corresponding b in the domain, such that the application of f from a through b comes at the specified expense. Applying this inductively with increasing lower bounds, the entire domain may be sectioned into any number of subdomains, all representing equal computational loads.

This technique results in an efficient algorithm for work transfer calculation, at the expense of restricting freedom of choice in the task selection phase. The effectiveness of the method is directly related to the fitness of the approximate cost function.

3 Experimental results

The method of balancing a computational load by integrating an approximating polynomial is applied to a hypothetical sample load, in order to provide a proof-of-concept implementation.

Two experiments are described in detail. These confirm the assumption about the relation between the fitness of the approximating polynomial and the effectiveness of the approach.

The first experiment approximates the load using a function which is integrated and solved with respect to its bounds analytically, in order to produce a work distribution. The resulting distribution is produced in a negligible amount of time, but the simplicity of the approximating polynomial creates a significant imbalance in the distribution, with some tasks requiring up to 20% longer execution time than others.

The second experiment utilises the Newton/Raphson iterative method for finding the bounds of the definite integral, which admits a better approximation of the load, but leads to greater complexity in the work transfer calculation. This improves the accuracy of the load division, resulting in a worst case of 8% longer runtimes for some tasks. The increased overhead for evaluating the bounds remains insignificant compared to the computation of the tasks themselves.

The applicability of a numerical solution affords greater liberty in the construction of the approximating polynomial, suggesting that still higher precisions may be attained without incurring an unreasonable overhead for computing transfer sets. In fact, the costs of evaluating the integrals and bounds which result in the work distributions of the experiments are small enough that difficulties arise when attempting to measure their magnitudes. Thus, the computational efficiency of the developed technique is considered sufficient for the method to be usable in interactive software without increasing response times noticeably.

4 Conclusions

The work demonstrates that it is possible to implement an algorithm which partitions a complex and nonuniform computational load into approximately equal parts with an almost nonexistent overhead.

These benefits come at the expense of not allowing the data set to alter during run time. It is also necessary to expend some amount of resources on profiling the data and function sets off-line, to facilitate the construction of appropriate polynomials which approximate the computational cost of applying the set of functions to the set of data.

References

1. Piersall, S., Elfayoumi, S.: DYLAPSI: A Dynamic Load-Balancing Architecture for Image Processing Applications. ISCA 15^{th} International Conference on Parallel and Distributed Computing (2002)
2. Watts, J., Taylor, S.: A Practical Approach to Dynamic Load Balancing. IEEE Transactions on Parallel and Distributed Systems **9(3)**, (1998) p. 235