A Universal Load Balancing System and its Application for the Isolation of Real Roots

Ph.D. Thesis

Thomas Decker

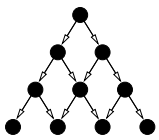
Abstract

The universal load balancing system "*Virtual Data Space*" (*VDS*) is the main focus of this dissertation. The system supports well-known load balancing techniques and new methods that are presented here in detail.

The dissertation consists of 7 chapters. It starts with a discussion of the scalability of parallel algorithms. Chapters 3 and 4 present new static scheduling algorithms, and Chapter 5 explores static load balancing of independent tasks. Chapter 6 overviews the main concepts of VDS. Finally, Chapter 7 shows how VDS is used to parallelize the Descartes method for polynomial real root isolation.

Chapter 2 presents a formal definition of the isoefficiency concept introduced by Kumar et al. Some fundamental properties resulting from the definition are derived. The definition makes it possible to analyze the isoefficiency of algorithms whose efficiency does not solely depend on the sequential computing time. The class of problems that can be solved by a parallel algorithm with polynomial isoefficiency turns out to be a strict sub-class of the class EP introduced by Kruskal et al.

Chapter 3 discusses static scheduling algorithms for the *pyramid dag*. A pyramid dag of height n consists of nodes $(i,\ j)$ where $0 \leq i,\ j < n$ and $i + j < n$. The computation associated with node $(i,\ j)$ needs results from the predecessors $(i-1,\ j)$ and $(i,\ j-1)$. Dependencies of this kind are typically generated by algorithms that make use of dynamic programming, such as triangulation of polygons. Applications from computer algebra include the complete Horner-scheme and the Taylor-shift. Scheduling algorithms compute a mapping of the dag nodes to processors.



First, lower bounds are given for the isoefficiency of pyramid dag computations. The isoefficiency function in the number P of processors is shown to be $\Omega(P^2)$ if all nodes have the same computing time. Even quite simple mapping schemes are optimal with respect to isoefficiency. However, the known mapping schemes are less efficient than *pie-mapping*, a new scheme that partitions the dag into contiguous zones. Doing so reduces the communication overhead from a quadratic dependency on the pyramid height to a linear one. This gain of efficiency becomes apparent in experiments even when the node computations are performed simply level-by-level. A further improvement is achieved by ordering the nodes within each zone according to a block scheme. The block order is shown to allow more time for communication.

Chapter 4 considers a different load balancing problem. A set of m independent tasks is given, and each task can be computed in parallel. The tasks are assumed to be uniform in the sense that they all have the same computing time function $t(p)$ in the number p of processors;

also, $t(p)$ is assumed to be known in advance. The problem can be stated as follows. Given a number P of processors and a computing time function $t(p)$, compute for each task a number of processors and a start time such that no processor computes more than one task at a time and such that the makespan is minimized. The most interesting situation arises when the tasks are outnumbered by the processors. For non-uniform computing time functions the problem is known to be NP-hard in the strong sense even for a fixed number of processors. For uniform tasks, an algorithm with a scheduling time of $O\left(m\,(2m+1)^{P^2-P}\,2^P\right)$ is presented. This bound can be reduced to $O\left(m\ (l\,t(1))^{P-1}\ 2^P\right)$ if the computing times are rational. Here, l is the smallest common multiple of $t(1),\ldots,t(P)$. Since the computing time grows exponentially in the number of processors this algorithm is worthless in practice. The problem is solved by introducing an approximation algorithm that computes *phase-parallel schedules* in time $\Theta(m^2)$. The makespan of an optimal phase-parallel schedule is at most twice the makespan of an optimal schedule.

Chapter 5 considers the distribution of independent tasks when tasks outnumber processors. The tasks are assumed to be uniform; initially, every task is assigned to one processor. The load balancing algorithm must re-distribute the tasks so that every processor is assigned the same number of tasks. This problem was studied for data-parallel applications in a dissertation by Diekmann. In that dissertation, the relationships between the tasks generate canonical neighborhoods of processors. Based on these neighborhoods, diffusion schemes are applied in order to balance the system load. The same techniques can also be used for independent tasks. However, due to missing task relationships there are no *a priori* neighborhoods. Hence, load balancing partners need to be defined by specifying a virtual topology. This chapter tries to identify topological properties that allow efficient load balancing. Two load balancing algorithms are considered; both are based on diffusion, and both operate in two phases. In the first phase, called *flow computation phase*, a balancing flow is computed; in the second phase, called *migration phase*, the load is re-distributed according to the computed flow.

An experimental evaluation of the two phases shows that topological properties such as the maximum degree and the number of distinct Laplacian eigenvalues influence the running time of the load balancing algorithm; so do properties of the balancing flow. While a balanced flow leads to a shorter migration phase it does require a longer flow computation. So, there is a tradeoff between the effort for flow computation and the effort for migration.

Chapter 6 gives a detailed description of the load balancing system VDS. Its underlying concept makes VDS universally applicable; indeed, VDS supports various programming paradigms and also the combination of different paradigms in the same application. A small example is provided to show how VDS applications are programmed. Comparing systems such as Cilk, Athapascan 1, and DOTS experimentally with VDS shows that VDS provides the most efficient support of strictly multithreaded computations on distributed memory machines.

Chapter 7 uses VDS to parallelize the Descartes method for polynomial real root isolation. The parallel Descartes method requires the combination of different programming paradigms—strictly multithreaded computations and static task graphs. The resulting parallel program is evaluated experimentally; its isoefficiency function is analyzed and shown to be $O(P^3 \log^2 P)$.