Abstract

Mass production, as an idea, has had a long and successful history. First recognized and distilled in the early 19th century by such men as Adam Smith and Charles Babbage, it has since been successfully applied to virtually every manufacturing process, from farming to automobile manufacture [1][2][3]. Strangely, though, these methods have not generally been considered to be applicable to the development (manufacture) of computer software. It is the thesis of this paper that not only would this be a good idea but that, in fact, such techniques are already being employed with high degrees of success. Four example applications, implemented on a massively parallel supercomputer, are presented to illustrate this point.

Introduction

It may at first seem that the normal images of mass production (car parts moving along a conveyor belt or mechanical robots designed to repeatedly tighten a single bolt) have nothing to do with software engineering or computer programming. Consider, however, that any programmer or group of programmers in a research lab seeking to solve a problem are producing a product. The product is the solution to the problem, and as with any industrial factory it is desired that this product be built as inexpensively and as efficiently as possible.

This paper describes how one of the rules of mass production - the use of powerful yet sometimes crude tools and processes - can be applied successfully to a variety of computer applications. The results allow for high quality solutions with a minimum of expense in person-years of development costs. This is accomplished through the use of simple, powerful but computationally expensive software methods such as simulated annealing [4], memory-based reasoning [5] and genetic optimization [6]; and is enabled by a computationally powerful massively parallel supercomputer [7].

The Rules of Mass Production

Mass production could be described as consisting of just the following two basic rules:
1. Tasks are specialized so that they can be carried out repeatedly in simple processes that require a minimum of wasted motion (either physical or computational).

2. Tasks and processes are modified to match the available tools.

For instance, farming has benefited greatly from the application of mass production techniques. Initially crops were cultivated by hand and farmers could sustain crops on small plots of land. Then a new tool, the hoe, was invented. It was crude, but it was a simple tool of wood and metal and readily available. Because it was cruder than doing things by hand, the process of farming changed to accommodate this new tool. Crops now had to be planted in long straight rows wide enough for the hoe (thereby using more land than was normally required to grow just the plants) and weeding and cultivating could only be performed within these rows - it was no longer possible to weed and till precisely between individual plants. Of course, now a single farmer could support several acres of crops.

There were many further stages in this evolution of farming tools, culminating in the large diesel powered backhoes and combines of today that enable a single farmer to cultivate hundreds of acres. Note that there are some general trends:

1). The precision of the tools decreased as waste from the process increased (not all the weeds were pulled, not all the soil was tilled).

2). The specialization of the tools increased (hands are very general tools, capable of playing the piano or opening jars, hoes are much more limited).

3). The power of the tools increased.

4). The process changed (farming now was accomplished in long, wide, straight rows).

5). The total production increased (from a fraction of an acre per day to many more).

Computer program development is in many ways like tending a farm. Here farming by hand is analogous to the programmer who makes use of few tools except perhaps the computer language he or she is writing in. The eventual product is very efficient, wastes no memory or CPU cycles, and is perfectly tailored to the task it was designed for. It is, however, very brittle in that small changes may require massive redesigns, and it may have required many person-years of effort to develop.

The mass production approach to computer program development attacks problems differently. Here a very powerful hardware tool is applied (a massively parallel supercomputer) and very computationally expensive software tools (genetic optimization, simulated annealing, and memory-based reasoning) are used to produce a system that performs well. Optimal use of computer memory and CPU cycles is not guaranteed but the system is built from generic tools and is not brittle. Small changes may be tried in order to optimize the system with little programmer cost; and the power of the hardware compensates for the losses of program efficiency. A tradeoff can, thus, be made between coarse, high level programming tools and program development time. This seems to be a