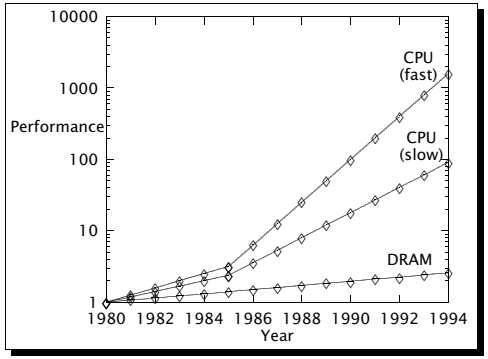

Figure 1: **CPU and memory performance.** This drawing uses 1980 as a baseline. Memory speed (dynamic random-access memory, DRAM) is plotted with an annual 7% increase. The slow CPU line grows at 19% annually until 1985, and at 50% annually since then. The fast CPU line rises at 26% annually until 1985, and at 100% annually since then. The data is taken from [73, Fig. 8.18, p. 427], but extended beyond 1992.

The existence of a memory hierarchy means that a few well-behaved programs will perform almost optimally on a particular system, but alas, most will not. Because the performance difference between the extremes of good and bad behavior can be several orders of magnitude, it is important for programmers to understand the impact of memory access patterns on performance.

Fortunately, once the issues are thoroughly understood, it is usually possible to control memory access in high-level languages, so it is seldom necessary to resort to assembly-language programming, or to delve into details of electronic circuits.

The purpose of these notes is to give the reader a description of the computer memory hierarchy, and then to demonstrate how a programmer working in a high-level language can exploit the hierarchy to achieve near-optimal performance.

2 1 INTRODUCTION

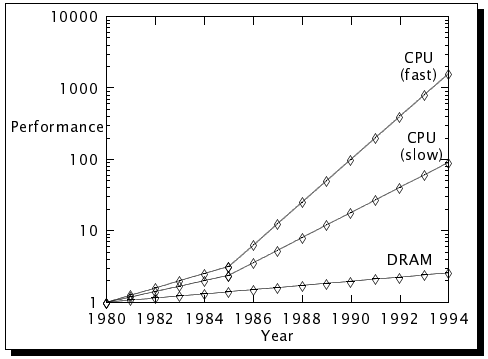

Figure 1: **CPU and memory performance.** This drawing uses 1980 as a baseline. Memory speed (dynamic random-access memory, DRAM) is plotted with an annual 7% increase. The slow CPU line grows at 19% annually until 1985, and at 50% annually since then. The fast CPU line rises at 26% annually until 1985, and at 100% annually since then. The data is taken from [73, Fig. 8.18, p. 427], but extended beyond 1992.

The existence of a memory hierarchy means that a few well-behaved programs will perform almost optimally on a particular system, but alas, most will not. Because the performance difference between the extremes of good and bad behavior can be several orders of magnitude, it is important for programmers to understand the impact of memory access patterns on performance.

Fortunately, once the issues are thoroughly understood, it is usually possible to control memory access in high-level languages, so it is seldom necessary to resort to assembly-language programming, or to delve into details of electronic circuits.

The purpose of these notes is to give the reader a description of the computer memory hierarchy, and then to demonstrate how a programmer working in a high-level language can exploit the hierarchy to achieve near-optimal performance.