Time-Memory-Processor Tradeoffs

Hamid R. Amirazizi and Martin E. Hellman

Department of Electrical Engineering

Stanford University

Summary

This paper demonstrates that usual time-memory tradeoffs offer no asymptotic advantage over exhaustive search. Instead, it proposes tradeoffs between time, memory, and parallel processing. Using this approach it is shown that most searching problems allow a tradeoff between C_s , the cost per solution, and C_m , the cost of the machine: doubling C_m increases the solution rate by a factor of four, halving C_s. The machine which achieves this has an unusual architecture, with a number of processors sharing a large memory through a sorting/switching network. The implications for cryptanalysis, the knapsack problem, multiple encryption and VLSI are discussed.

To formalize, we assume that c_p is the cost of one processor, c_m is the cost of one word of memory (or bit, since constant factors are neglected), and P and M are the number of processors and words of memory used by the machine, so that

$$C_m = c_p P + c_m M \tag{1}$$

But c_p and c_m are just constant factors, which we are neglecting so we can simplify (1) to

$$C_m = \max(P, M) \tag{2}$$

Equations (1) and (2) assume that memory and processors are the dominant cost of the machine. Later, when we add additional components (e.g., a switching network so any processor can access any word of memory), we must ensure that their costs do not dominate the cost of the machine by more than

constant or logarithmic factors which we are neglecting.

Using (2), C_r, the cost per run, is

$$C_r = C_m \cdot T = \max(P,M) \cdot T \tag{3}$$

and C_s, the cost per solution, is

$$C_s = C_r / S = \max(P,M) \cdot T / S \tag{4}$$

where S is the simultaneity of solution (the number of problems solved simultaneously in one run).

In the following we shall find the various (C_m, C_s) points.

1. Exhaustive search has $P=M=1$, $T=N$, $S=1$ so $C_m=1$ and $C_s=N$.
2. The usual form of table lookup has $P=1$, $M=N$, $T=1$ (neglecting precomputation) and $S=1$ so $C_m=N$ and $C_s=N$.
3. Usual time-memory tradeoffs have $P=1$ so $C_m = \max(P,M)=M$. Because $S=1$ and $MT=N$ it follows that $C_s=N$.

Under these operating conditions time-memory tradeoffs have no advantage over exhaustive search or table lookup. They have the same cost per solution as exhaustive search, but require more expensive machines. While they make up for the increased C_m by obtaining solutions more rapidly (for fixed C_s, T is inversely proportional to C_m), exhaustive search can do the same by recourse to parallelism (using P parallel processors increases C_m and decreases T by a factor of P so that C_s is unchanged)..

General Searching Problem TMP Tradeoff

In this section we shall develop an algorithm for solving the following problem : Given K values, $Y^{(1)}, \ldots, Y^{(K)}$, find the corresponding $X^{(1)}, \ldots, X^{(K)}$, such that $Y^{(K)}=f(X^{(K)})$. We allow f to be an arbitrary mapping from the set $\{1,\ldots,N\}$ to a set of arbitrary size. For example, in cryptography f may be a mapping from N keys to one of L messages. Neglecting the constant and logarithmic factors our proposed architecture and algorithm allows a TMP tradeoff with the

following parameters : $C_m = K$, $S = K$, $T = N/K$, and $C_s = N/K$ for $1 \leq K \leq N$. Note that $C_m \cdot C_s = N$ and therefore we have a tradeoff between C_m and C_s,i.e., increasing the cost of the machine by a factor c increases the solution rate by c^2, and thereby decreasing the cost per solution by a factor of c.