COMPUTING WITH MULTIPLE MICROPROCESSORS

John V. Levy
Stanford University

Computer systems with multiple processors are becoming more common for reasons of reliability and modularity. The use of asynchronous processors, however, leads to problems of complexity of control and of programming. This work investigates the application of multiple asynchronous processors to the computing task at the lowest level - that of interpreting single machine-language instructions.
A particular computer configuration with 15 identical processors has been constructed using an interpretive simulator. The processors are of relatively low computing capacity. A common data bus connects the processors with each other and with the main memory. A restriction on the logical connections between processors allows each one to communicate with no more than 2 others, in a chain-like arrangement. Three examples - 2 sort instructions and a matrix multiply - were coded for this machine and run using the simulator. By varying the bus cycle time, it was concluded that adequate support of up to 15 processors can be provided by a common bus with cycle time equal to the processor cycle time.
The amount of parallelism achieved was significant but showed dependence on hardware parameters and on the algorithm implementations. Direct simulation the the computer, with an execution trace of the running system, has yielded some glimpses of how restriction of bus capacity can cause deterioration of the program execution efficiency and amount of parallelism.

A simple economic model of a multiple processor system is developed and applied to the 3 examples. The result shows that the minimum cost per throughput occurs with 4,11, and 15 processors, respectively, for the 3 examples when the cost of a processor is 1/10 of the system cost.

THE ANALYSIS AND SCHEDULING OF DEVICES HAVING ROTATIONAL DELAYS

Samuel Henry Fuller
Stanford University

A number of problems concerning the scheduling, organization, and configuration of auxiliary storage units are analyzed in this dissertation. Stochastic, combinatorial, or simulation techniques are applied, depending on the assumptions and complexity of the particular problem. For the relatively simple scheduling disciplines of first-in-first-out (FIFO) and shortest-latency-time-first (SLTF), stochastic models are used. The starting addresses of I/O requests to a file (non-paging) drum are modeled as random variables that are uniformly distributed about the circumference of the drum; the lengths of I/O requests are modeled as random variables that are exponentially distributed. This model of I/O requests is based upon measurements from an operational computer system. The arrival times of I/O requests are first modeled as a Poisson process and then generalized to the case of a computer system with a finite degree of multiprogramming. Well-known results in queueing theory are sufficient for some models, but in other cases original approaches are required. In particular, a new model of the SLTF file drum is developed, is compared with previous models of the SLTF file drum as well as a simulation model, and is found to be a more accureat model than previously available. Another practical problem that is discussed is an I/O channel serving several, asynchronous paging drums. A new scheduling discipline is presented to minimize the total amount of rotational latency (and processing time) for an aribtrary set of N I/O requests and the algorithm that is developed to implement this minimal-total-processing-time (MTPT) scheduling discipline has a computational complexity on the order of NlogN. The MTPT scheduling algorithm was implemented, and for more than three or four records, the most time-consuming step is the initial sorting of the records, a step also present in SLTF scheduling algorithms.

PERFORMANCE EVALUATION OF MULTIPROGRAMMED TIME-SHARED COMPUTER SYSTEMS*

Akira Sekino
MIT

A comprehensive set of hierarchically organized modular analytical models is developed for performance evaluation of multiprogrammed virtual-memory time-shared computer systems. This hierarchy of models includes a user behavior model, a secondary memory model, a program behavior model, a processor model, and a total system model. The thesis especially details on the last three models. The program behavior model permits estimation of the frequency of paging. The processor model evaluates the throughput of a given multi-processor multi-memory processing system under multiprogramming. Finally, the total system model derives response time distribution of an entire computer system under study. Accuracy of performance prediction by these models is examined by comparing the predicted performance and the measured performance of the Multics system. These analyses are then applied to the optimization of computer systems and to the selection of the best performing configuration for a given budget. This framework of performance evaluation not only guides human intuition in understanding actual performance problems but presents reliable answers to quantitative performance questions about throughput and response time of actual computer systems.

* Available from M.I.T. Project MAC as MAC-TR-103.

THE EFFECT OF SELECTED VARIABLES ON THE LEARNING OF SOME COMPUTER PROGRAMMING ELEMENTS

Sorel Reisman
The Ontario Institute for Studies in Education

The widespread use of computers in most aspects of society has caused educators to introduce computing into many levels of the curriculum. In the absence of experience suggesting techniques for structuring such courses, an intuitive approach is usually taken. A conflict which has arisen because of this concerns the advantages or disadvantages of teaching high level languages before low level languages. This study attempted to resolve this conflict. An additional objective of the study was to investigate the effects of providing learners with a) computer generated reinforcement and b) differential error diagnostics in their programming exercises.

Elements of high level and low level programming languages were selected to be illustrative of programming languages taught in Ontario secondary schools. Two programmed instruction (PI) manuals, each representing one of the language levels, were written and field-tested for this study. Two sets of problems, one for each manual, were designed to be administered in two interactive time-sharing sessions.

The main study was carried out in two phases. The first phase was conducted on 42 graduate students and the second on 113 grade ten students. The treatments to which each S was randomly assigned were a) high level/low level or low level/high level; b) error code or error statement; and c) reinforcement or no reinforcement. Depending on the sequence which had been assigned, the first PI booklet was administered to each S. Following this, each S worked on programming problems administered via a time-sharing terminal where appropriate error and reinforcement diagnostics were generated. A second PI manual was administered to S followed by a second time-sharing session. At the completion of these sessions, a posttest was administered to all Ss.

Performance data collected during the Ss' time-sharing sessions, and various partial posttest measures, were analysed with an analysis of covariance. Some significant differences and strong trends tended to support the hypothesis that the low level/high level sequence results in better overall performance than the reversed sequence.

A questionnaire distributed to all grade ten Ss indicated that they preferred the low/high sequence and thought it to be less difficult than the high/low sequence.

No significant differences or strong trends were evident as a result of the error or reinforcement treatments.

The results of sequence difference, while appearing small in this study, seemed quite distinct. This suggests that these differences either exist and are of no practical value to the practitioner, or that they might become more distinct in a full term computing course.

The error and reinforcement treatments which did not result in differentiating performance tend to support the findings of other studies done in interactive computing environments. That is, programmers working on time-sharing terminals appear to ignore programming manuals and computer-generated assistance, and seem to try different strategies when they make errors. This suggests that the treatments used in this study might produce different results for learners using batch computer systems.

Computational Experience With The Central Cutting Plane Algorithm

Jack Elzinga, Johns Hopkins University and
Thomas G. Moore, Johns Hopkins University

The Central Cutting Plane Algorithm is an algorithm for solving the convex programming problem. Its convergence and other properties have been established elsewhere [2,6]; here, we will state, but not prove, the major theoretical results concerning the algorithm. The purpose of this presentation is to report some preliminary computational experience with an experimental code of the algorithm.

THE ALGORITHM

The Central Cutting Plane Algorithm (CCPA) is designed to solve the following form of the convex programming problem:

$$\text{maximize} \quad c^t x$$

$$\text{s.t.} \quad g_i(x) \geq 0 \quad i=1,\ldots,m \qquad \text{(NLP)}$$

$$x \varepsilon S \subset E^n.$$

where the constraint functions g_i are assumed to be continuously differentiable, concave functions, S is a compact, convex set, and c is a fixed vector in E^n. The assumption of a linear objective function involves no loss of generality since a problem where the objective function is a continuously differentiable concave function, not necessarily linear, may be transformed into a problem with the structure of (NLP).

The Central Cutting Plane Algorithm is as follows (Let $I = \{i \mid 1 \leq i \leq m\}$, and $F = \{x \mid g_i(x) \geq 0 \ \forall i \varepsilon I, \ x \varepsilon S\}$) :

Step 0. Select $x^o \varepsilon S$. If $x^o \varepsilon F$, $f^o = c^t x^o$. If $x^o \notin F$, let $f^o = -\infty$. Let $z^o = w^o = x^o$, $J_i^o = \phi$ for $i \varepsilon I$, β be a fixed scalar, $0 \leq \beta < 1$, and $k=0$.

Step 1. Let (x^{k+1}, σ^{k+1}) be the solution to the mathematical programming problem, SP_k:

$$\max \sigma$$

$$\text{s.t.} \quad cx - \|c\| \sigma \geq f^k$$

$$g_i(y) + \nabla g_i(y)(x-y) - \|\nabla g_i(y)\| \sigma \geq 0 \quad y \varepsilon J_i^k, \ i \varepsilon I$$

$$x \varepsilon S.$$

If $\sigma^k = 0$, stop. Otherwise, go to Step 2.

Step 2 i) If $x^{k+1} \varepsilon F$, let z^{k+1} be any point such that $cz^{k=1} \geq cx^{k+1}$. Determine $w^{k+1} = \rho z^{k+1} + (1-\rho) w^k$ for $0 \leq \rho \leq 1$, such that $g_r(w^{k+1}) \leq 0$ for some $r \varepsilon I$.

ii) If $x^{k+1} \notin F$ and $z^k \notin F$, let $w^{k+1} = x^{k+1}$. Determine $r \varepsilon I$ such that $g_r(w^{k+1}) \leq 0$.

iii) If $x^{k+1} \notin F$, and $z^k \varepsilon F$, determine $\hat{z}^{k+1} \varepsilon F$ such that $c\hat{z}^{k+1} \geq cz^k$. Determine z^{k+1} and w^{k+1} such that $z^{k+1} \varepsilon F$, $z^{k+1} = \tau \hat{z}^{k+1} + (1-\tau) x^{k+1}$, and $g_r(w^{k+1}) \leq 0$ for some $r \varepsilon I$, and $w^{k+1} = \rho \hat{z}^{k+1} + (1-\rho) x^{k+1}$, $0 \leq \rho \leq \tau \leq 1$.

Step 3. If there exists any point(s) w^p such that $p \leq k$, $w^p \varepsilon J_i^k$, $i \varepsilon I$, and

i) the constraint $g_i(w^p) + \nabla g_i(w^p)(x - w^p) - \|\nabla g_i(w^p)\| \sigma \geq 0$

is not binding in SP_k, and

ii) $\sigma^k \leq \beta \sigma^p$

then let $J_i^k = J_i^k - w^p$. After this process has been carried out, let $J_i^{k+1} = J_i^k$ for all $i \varepsilon I$, $i \neq r$.

Step 4. Let $f^{k+1} = cz^{k+1}$, and let $J_r^{k+1} = J_r^k \cup \{w^{k+1}\}$. Set k=k+1 and repeat from Step 1.

The subproblem SP_k has the property that x^{k+1} will be the center, and σ^{k+1} the radius, of the largest hypersphere which can be inscribed inside the polyhedron $\{x \mid cx \geq f^k, g_i(y) + \nabla g_i(y)(x-y) \geq 0, \forall y \varepsilon J_i^k, i \varepsilon I\}$ with the further restriction that x^{k+1} must lie in S. Thus x^{k+1} is the "center" of a polyhedral approximation to the solution set of NLP. Hence, the name of the algorithm, the Central Cutting Plane Algorithm. If $x^{k+1} \varepsilon F$, then $f^{k+1} \geq cx^{k+1}$, and if $x^{k+1} \notin F$, then $g_i(w^{k+1}) + \nabla g_i(w^{k+1})(x^{k+1} - w^{k+1}) \leq 0$. Since $\sigma^{k+1} \geq 0$ for all k, x^{k+1} is effectively cut off by these additional constraints forming SP_{k+1} from SP_k. (Cuts of the form $g_i(y) + g_i(y)(y-x) \geq 0$ will be termed "constraint cuts", while cuts of the form $cx \geq f^k$ will be termed "objective cuts.")

Step 2 allows a great deal of flexibility in the determination of z^{k+1}, w^{k+1}, and $\hat{z}^{k+1}$. With one additional assumption concerning S, namely, that there exists a point $\hat{x} \varepsilon S$ such that $g_i(\hat{x}) > 0$ for all $i \varepsilon I$, it can be shown that all accumulation points of the sequence $\{z^k\}_{k=1}^{\infty}$ are optimal solutions to NLP, regardless of the specific method of computing z^{k+1}, w^{k+1}, $\hat{z}^{k+1}$ to satisfy the conditions of Step 2.

The purpose of Step 3 is to keep the number of constraints in SP_k small. Every time a new w^{k+1} is determined, another constraint is added to the subproblem. The rule given in Step 3 for dropping constraints preserves the convergence properties of the algorithm.

In practice the set S will be a bounded polyhedron described by linear constraints, so SP_k will be a linear programming problem. The subproblems are then solved by solving the dual of SP_k; then the addition of new constraints to SP_k corresponds to adding new columns to the dual problem.

The flexibility in Step 2 allows for various techniques to accelerate the algorithm. The following scheme represents no acceleration: if $x^{k+1}\epsilon F$, let $z^{k+1} = x^{k+1}$, $\rho=0$; if $x^{k+1}\notin F$, $z^{k+1} = \hat{z}^{k+1} = z^k$, $w^{k+1} = x^{k+1}$ in iii). We will report on a number of experimental acceleration techniques.

Acceleration Technique I

If the center, x^{k+1}, found by the subproblem is feasible, no other technique is employed to accelerate the algorithm. The addition of an objective cut (that is, changing the right hand side of the objective cut, or, in the dual problem, changing the cost associated with that column) may require no pivots in the linear program; thus this improvement in f^k is gained with little computational effort.

If the center, x^{k+1}, is infeasible, then a bisecting search is carried out on the line segment between x^{k+1} and z^k. That is, the point $\frac{1}{2}(x^{k+1}+z^k)$ is evaluated to determine whether or not it is feasible. If it is infeasible, then it is used instead of x^{k+1} as the most recently found infeasible point; if feasible, it replaces z^kas the most recently found feasible point. The result of this is that a feasible and an infeasible point are still determined, but the distance between them is half the distance between the starting feasible and infeasible points. This procedure can be repeated a certain number of times with the final feasible point, z^{k+1}, and the final infeasible point, w^{k+1}, being used to generate the new objective cut and constraint cut. The number of bisecting steps used was $[n/m] + 1$, where $[x]$ indicates the largest integer less than or equal to x. The reason for this rule is that the value of this acceleration step is dependent on the number of constraints. The most likely numbers of function evaluations required to determine whether the new midpoint of the segment is feasible or not are 1 or m_1; if it is infeasible, it is likely to be infeasible to the same constraint as the infeasible point determining the end-point of the segment and can thus be identified with one function evaluation by evaluating this constraint first; if it is feasible, all m_1 constraints may have to be evaluated to determine this. Thus, the value of the bisecting step with respect to its cost in terms of computational effort is inversely related to the number of nonlinear constraints in NLP.

Acceleration Technique II

A second acceleration technique was used on some of the test problems. This technique consists of three consecutive line searches if the center x^{k+1} is infeasible; as in the other acceleration technique, no additional action is taken if x^{k+1} was feasible. The first search direction is determined by projecting the direction from z^k to x^{k+1} onto the hyperplane $cx=cz^k$, i.e. $d=(z^k-x^{k+1}) - [(z^k-x^{k+1})^t c/\|c\|^2]c$. Then the function min $[\theta_i g_i(x),\ i\ldots,m]$, where θ_i are approximate scale factors for the functions, is maximized along this line. The next direction is in the direction c, moving as far as po-sible without leaving the feasible set. This determines $\hat{z}^{k+1}$. Since $\hat{z}^{k+1}$ is close to the boundary, a small step in the direction of x^{k+1} then yields an infeasible point for w^{k+1}. The purpose of the first step is to get away from the constraints; the purpose of the second is to find a better feasible point; the purpose of the third is to find an infeasible point close to the boundary from which to generate the constraint cut. Since it is not necessary to find the exact maximum of the function in the first step, nor is it necessary to exactly determine the boundary in the second or third step, only a few evaluations need be performed in each of the three steps. When the linear constraints describing S play an active role, this acceleration technique must be modified, in an obvious fashion, by employing projection operations onto the active linear constraints.

Acceleration Technique III

This technique involves a line search along a segment of what is termed the "central ray" of the polyhedral approximation to the optimal solution set. The central ray is constructed (and defined) by taking the optimal solution of a subproblem and performing the following: first, all inactive constraints (cuts) are discarded. Then, the objective function value determining the objective cut is allowed to vary from $-$ to $+$. The locus of all centers resulting from this perturbation is the central ray. By this construction, the central ray is seen to be the ray passing through the center of the polyhedron and the point obtained if the objective function were maximized over the polyhedron described.

A generator for the central ray by the active cuts is available from the optimal basis inverse in the dual problem. The line search for the boundary is then carried out between the center and the upper or lower bound contour depending whether the center was feasible or infeasible.

It may appear that the insertion into CCPA of the line searches described in the acceleration techniques robs the algorithm of one of its attractive features. It should be emphasized once again however that the convergence of the algorithm does not depend on the exactness of the line searches (in fact, employing no line searches at all results in a convergent algorithm); therefore, the type of acceleration technique employed and the effort expended on it are decisions that can be made on numerical or heuristic considerations.

COMPUTATIONAL RESULTS

Bracken and McCormick [1] relate the following nonlinear programming problem:

$$\min \sum_{j=1}^{n} c^j x^j \qquad (1)$$

$$\text{s.t. } \sum_{j=1}^{n} a_{ij}x_j - 1.645\left(\sum_{j=1}^{n} v_{ij}x_j^2\right)^{\frac{1}{2}} \geq b_i \quad i=1,\ldots,m \qquad (2)$$

$$\sum_{j=1}^{n} x_j = 1 \qquad (3)$$

$$x_j \geq 0,\ j=1.,,,.m. \qquad (4)$$

The purpose of the problem is to determine the optimal feed mixture for livestock. The variable x_j represents the fraction of feed j to be included in the mixture, c_j is the cost per unit of feed j, a_{ij} is the average amount of some attribute i (e.g. protein content) per unit of feed j, v_{ij} is the variance of the amount of attribute i per unit of the mixture. If the level of attribute i is independently normally distributed for each feed, then (5.2) represents the constraint that the mixture should have a 95% probability of attaining the desired level of attribute i. Thus the problem is to minimize the cost of the mixture while maintaining a 95% probability for attaining the desired level for each attribute.

The constraint function in (5.2) is concave. Therefore, this is a convex programming problem. The problem is a good problem for testing purposes. Problems of any size can be generated by generating appropriate a_{ij}'s, v_{ij}'s, and b_j's. The square root makes function evaluations non-trivial.

In order to test the efficiency of the algorithm, data were generated for problems of varying sizes. The parameters were chosen randomly according to the following scheme: a_{ij} was uniformly distributed over [0, 20], v_{ij} over [0, 7], c_j over [0, 10], and b_i was determined by the formula $b_i = \bar{b}_i - (\bar{b}_i - \hat{b}_i)(1/.95)(\bar{r})$, where $\bar{b}_i$ was the value of $\sum_{j=1}^{n} a_{ij}x_j - 1.645(\sum_{j=1}^{n} v_{ij}x_j^2)^{\frac{1}{2}}$ evaluated at $x_j = \frac{1}{n}$ for all j, and $\hat{b}_i = \min_j[a_{ij} - 1.645(v_{ij})^{\frac{1}{2}}]$ and $\bar{r}$ was a random variable uniformly distributed over [0, 1]. In this way, the point $x_j = \frac{1}{n}$ for all j was feasible to every constraint, but there was also a 95% probability that each constraint cut off at least one extreme point of the polyhedron described by (3) and (4).

Since the computation time required to solve a problem is dependent on the computer used, the problems generated were also solved using the SUMT algorithm (Fiacco and McCormick [3]) to provide a measure of the difficulty of the problems. The 1964 version was used. Some experimenting was done on trial problems to determine a good set of options and parameter values for the SUMT algorithm on this class of problems. The initial value of r was determined by the formula $r = [(\nabla f(x^\circ)^t H^{-1}(x^\circ)\nabla f(x^\circ))/(\nabla B(x^\circ)^t H^{-1}(x^\circ)\nabla B(x^\circ))]$ where $B(x) = \sum_{i=1}^{\bar{m}} 1/g_i(x)$, H is the hessian of B, f is the objective function and $\bar{m}$ is the number of constraints. This appeared to give better results than the standard r=1. The ratio of decrease for r was $\frac{1}{4}$, and second order extra-polation moves were employed. A good subproblem stopping rule was to stop when the inner product of the search direction with the gradient of the subproblem function was less than 10^{-1} times the tolerance desired to stop the entire algorithm (i.e. the difference between the upper and lower bounds on the optimal objective function value).

Additionally, the CCPA code was modified to perform the cutting plane algorithm of Kelley [5] (KCPA). The algorithm was stopped when a subproblem solution fell within the tolerance of the best upper bound determined by SUMT and CCPA. Knowledge of such a bound is advantageous to KCPA since it generates only upper bounds itself.

Special consideration had to be given to the constraint (3) since neither algorithm was designed for equality constraints. For SUMT, the variable x_n was replaced everywhere by $1 - \sum_{j=1}^{n-1} x_j$ and the constraint $1 - \sum_{j=1}^{n-1} x_j \geq 0$ replaced $x_n \geq 0$. For CCPA and KCPA, (3) was replaced by $\sum_{j=1}^{n} x_j \geq 1$ and $-\sum_{j=1}^{n} x_j \geq -1$.

In Figure 1, the solution times are shown for SUMT, KCPA, and two versions of CCPA, the faster times belonging to an accelerated version (acceleration technique I), and the slower times belonging to a version without any acceleration technique. Problems of various sizes were solved to determine the effect of problem size on computation time. The algorithms were terminated when the difference between the upper and lower bounds on the objective function was .001. (In CCPA, f^k is the lower bound, and an upper bound can be easily calculated from the subproblem: $cz^* \leq f^{k+1} + \sigma^{k+1}/u^k$, where u^k is the dual variable corresponding to the objective cut constraint in SP_k.) The time is computation time only (i.e. input and output times are not included). The problems were solved on an IBM 7094. As evidenced from these computational results, the Central Cutting Plane Algorithm appears to be quite efficient.

Certainly one of the strong features of the cutting plane algorithms are their ability to handle linear constraints routinely. Although at first glance these test problems appear to have very few linear constraints, in fact there are quite a few -- each of the non-negativity constraints is handled very routinely by the CCPA and KCPA since they solve linear programming subproblems. This factor is responsible in part for the encouraging computational results. Since one would expect a certain number of linear constraints to appear in most nonlinear programming applications, the Central Cutting Plane Algorithm seems particularly attractive.

In order to remove this linearity from the test problems, the constraints (3) and (4) were replaced by

$$\sum_{j=1}^{n} (x_j - 1)^2 \leq 1. \qquad (5)$$

Thus the solution was restricted to lie within a ball of radius 1 around the point (1,1,...,1). Test problems were generated in the same manner as before except that the b_i were determined so that (1,1,...,1), rather than (1/n,...,1/n), was the starting feasible point. The function

$$\sum_{j=1}^{n} a_{ij} x_j - 1.645(\sum_{j=1}^{n} v_j x_j^2)$$

was evaluated at (1,1,...,1) and either $(1 + 1/\sqrt{n}, \ldots, 1 + 1/\sqrt{n})$ or $(1 - 1/\sqrt{n},\ldots,1 - 1/\sqrt{n})$ depending on the sign of the evaluation at (1,1,...,1). Then a random fraction, uniformly distributed on $[\frac{1}{2}, 1]$, times the difference in these evaluations was subtracted from the evaluation at (1,1,...,1) to determine b_i, thus ensuring that at least some points feasible to (5) were not feasible to any constraint generated. The a_{ij}'s were randomly selected from [0, 20] and the c_j's from [0, 10], as before. The v_{ij}'s were randomly selected from [0, 50n].

The problem was solved exactly as formulated using SUMT. For CCPA the non-negativity constraints were included, plus the constraints $\sum_{j=1}^{n} x_j \geq n - \sqrt{n}$ and $-\sum_{j=1}^{n} x_j \geq -n - \sqrt{n}$. These constraints had no effect on the solution since the ball described by (5) lay within these constraints. However, they provided a compact set S for CCPA, and also bounded the solution away from (0,0,...,0) where the gradients were undefined.

The computation times for these problems are shown in Figure 2. The computations were terminated when the difference between the upper and lower bounds on the objective function was less than .001 times the value of the objective function at (1,1.,,,.1). The acceleration technique used in CCPA was the second technique discussed in this chapter. A few problems with 40 variables were attempted, but were determined to require too much computation time (over 5 minutes, at least for problems with only a few constraints) to be feasible for study. Note that the times for SUMT are approximately half of that recorded for problems of comparable size in Figure 1; essentially this is due to the dropping of the non-negativity constraints, which effectively reduces the number of constraints by half.

The Central Cutting Plane Algorithm, although performing acceptably, did not solve this class of problems with the speed exhibited earlier. This is due to the difficulty in approximating nonlinear curv-s by linear approximations. Note that the computation time decreased with an increase in the number of constraints. This was because with a large number of constraints the optimum was more likely to occur at a sharply defined vertex of the feasible region, where several nonlinear constraints intersected; with fewer constraints, the optimum was more likely to lie on a smoother boundary of the feasible region. The linear approximations generated by CCPA are much better at approximating a sharp vertex than at approximating a smooth curved surface.

The chemical equilibrium problem, which minimizes the Gibbs free energy subject to material balances, is a convex programming problem with linear equality constraints and non-negativity constraints. We have solved the problem given by Bracken and McCormick [1], which, after formulating an equivalent problem with a linear objective, results in a problem with 11 variables, 1 nonlinear constraint, 3 linear equalities and non-negativity constraints on all variables. The problem is complicated by the non-differentiability of the free energy function at zero. This difficulty is handled by treating the non-negativity constraints as nonlinear constraints; then finding the center of a polyhedron defined in part by these non-negativity constraints constrains all solutions away from the troublesome zero values. A solution optimal within the tolerance 0.001 was obtained in 28 seconds by CCPA; the computation time was dropped to 18 seconds using the central ray accelerator (CCPA-III). Acceleration technique I did not perform well on this problem, creeping slowly along the nonlinear constraint until some numerical difficulties occurred (In fact, this phenomonon led us to devise acceleration techniques II and III which also seek better feasible-infeasible pains, but restrict that search to the "middle" of the polyhedral approximation). These times compare favorably with other reported solution times [4] on more modern machines. Direct comparison is not possible since tolerances and stopping rules are not reported.

The close similarity between the chemical equilibrium problem and the geometric programming dual problem suggests that the latter might profitably be attacked by CCPA. Part of CCPA's attractiveness for use on the geometric programming dual stems from the natural way it handles the problem of non-differentiability.

DISCUSSION

The implementation presented here has the attractive feature that no changes need to be made to the linear programming code. The CCPA code for which the computational results were reported used the SIMPLX code written by George Diderrich of the University of Wisconsin Computing Center and modified by Al Kacala of The Johns Hopkins University Computing Center for use on the IBM 7094.

Certainly a more efficient code could be designed for the CCPA. Not only might the algorithm be improved by the design of other acceleration techniques, but even the code reported on here could be improved by programming a specialized linear programming code for the Central Cutting Plane Algorithm. For example, in using the CCPA, it is known in many cases which column is the prime candidate to enter the basis (the new cut), and so savings could be made in pricing out the columns. Also, features could be incorporated to allow dropping the non-negativity restriction on the dual variables.

Nevertheless, it is encouraging to have the assurance that the addition of a few subroutines to an efficient linear programming code can yield, particularly when linearities are present in the nonlinear problem, an efficient code for the convex programming problem.

REFERENCES

1. Bracken, J., and G. P. McCormick, Selected Applications of Nonlinear Programming, John Wiley and Sons, Inc., New York, 1968.

2. Elzinga, Jack, and Thomas G. Moore, "The Central Cutting Plane Algorithm," to appear.

3. Fiacco, A. V., and G. P. McCormick, "Computational Algorithm for the Sequential Unconstrained Minimization Technique for Nonlinear Programming," Management Science 10, 601-617 (1964).

4. Himmelblau, David M., Applied Nonlinear Programming, McGraw-Hill Book Co., New York, 1972.

5. Kelley, J. E., Jr., "The Cutting-Plane Method for Solving Convex Programs," Journal SIAM 8, 703-712 (1960).

6. Moore, Thomas G., "The Central Cutting Plane Algorithm," (unpublished Ph.D. dissertation. Department of Mathematical Sciences, The Johns Hopkins University, 1973).

VARIABLES	NONLINEAR CONSTRAINTS	SUMT	KCPA	CCPA	CCPA-I
5	5	4.2	0.6	0.6	0.5
10	5	8.9	0.6	0.6	0.6
10	10	14.2	1.1	1.6	1.5
10	20	26.3	3.6	3.0	3.0
20	10	45.0	6.4	4.5	3.7
20	20	77.2	8.6	6.8	6.2
20	40	130.0	26.3	15.8	14.1
40	40	450*	52.0	45.3	35.9
60	60		411.5	269.1	128.2

*problem terminated before completion

Figure 1. Computation Time (IBM 7094) in Seconds for the Feed Problem.

VARIABLES	NONLINEAR CONSTRAINTS	SUMT	KCPA	CCPA-II
5	2	0.9	3.9	0.9
5	4	1.9	3.4	2.1
5	5	2.2	3.5	1.3
5	10	4.2		2.2
10	4	2.9		9.8
10	7	5.2		9.3
10	10	7.3		8.4
10	20	13.3		7.9
20	7	12.1		128.1
20	14	22.2		85.2
20	20	33.4		93.6
20	40	66.2		77.3

Figure 2. Computation Time (IBM 7094) in Seconds for the Modified Feed Problem