Interval Computations
No 2, 1991

PARALLEL COMPUTERS. ESTIMATE ERRORS CAUSED BY IMPRECISE DATA

Vladik Kreinovich, Andrew Bernat,
Elsa Villa, Yvonne Mariscal

A typical usage of computers in engineering is as follows: we are interested in some quantity y (e.g., the amount of oil in a given region), that is difficult or even impossible to measure directly. So we measure some physical quantities $x_1, \ldots, x_n$, that are connected with y (e.g., conductivity of different layers) and then compute y from the x_i. Measurement errors in x_i lead to an error d in y. The problem is to estimate d, but methods of numerical analysis are often too difficult.

It is appropriate to consider solution of this problem within the context of software engineering because we are essentially concerned with the quality of the results of our programs. We show that solution of this problem is feasible, that is, it is possible to write a program that computes both y and its error estimate d from x_i, given the measurements x_i and the range of possible measurement errors.

We show how this method can be parallelized and what parallelization model fits this problem best.

ПАРАЛЛЕЛЬНЫЕ КОМПЬЮТЕРЫ. ОЦЕНИВАНИЕ ПОГРЕШНОСТЕЙ, ВЫЗЫВАЕМЫХ НЕТОЧНЫМИ ДАННЫМИ

Владик Крейнович, Эндрю Бернат,
Эльза Вилла, Ивонн Марискал

Обычно компьютеры в технике используются следующим образом: мы интересуемся некоторой величиной y (например, количеством нефти в заданном регионе), которую трудно или даже

невозможно измерить непосредственно. Для этого мы измеряем физические величины $x_1, \ldots, x_n$, которые связаны с y (например, проводимость различных слоев) и вычисляем y по x_i. Ошибки измерений x_i порождают ошибку δ для y. Нашей задачей является оценить d, но методы численного анализа часто оказываются слишком сложными.

Представляется разумным рассмотреть решение этой задачи в контексте построения средств вычислительной техники, т.к. по существу мы заинтересованы в качестве результатов наших программ. Мы покажем, что решение задачи может быть найдено; так возможно написать программу, которая вычисляет и y и оценку ошибки d по x_i и по пределам возможных ошибок измерений.

Мы покажем, как этот метод может быть распараллелен и какая модель распараллеливания соответствует нашей задаче наилучшим образом.

1. Introduction

A typical usage of computers in engineering is as follows. We are interested in some quantity y (e.g., the amount of oil in a given region), and it is difficult (or even impossible) to measure y directly. So we measure some physical quantities $x_1, \ldots, x_n$ that are indirectly connected with y (e.g., the conductivity of different layers, the results of ultrasound screening, etc.) and then compute y by applying some known algorithm to these values x_i. This algorithm is actually the numerical method for solving the equations that connect y with x_i (these equations can be algebraic, integral, differential, stochastic etc).

The program that solves these equations numerically gives us some value of y, e.g., in the oil example $y = 13$, meaning that there are 13 million tons. But the measurements are inevitably not precise; so this "13" is also not precise. An important question for a user is: What is the possible error of this estimate? If the maximum possible error is 1 then this estimate is quite reasonable, but if the error caused by the imprecise data can be as large as 100, then this estimate is senseless.

The traditional approach to answering this question is to analyze the corresponding mathematical model. Engineers who have gone through numerical mathematics know very well that this is often an extremely

difficult problem. Even for the simplest case, when y and x_i are related by a system of a linear equations, error estimates are difficult to compute, and for some nonlinear systems error estimates are not even known. So these methods are hard to use, and for an engineer, who is not a specialist in numerical analysis, it is a very tough (and sometimes even impossible) job.

The ideal solution for an engineer is when the computer itself will compute these errors. That is, the user chooses (or writes himself) a program that computes y from x_i, tells the computer the results x_i, : $i = 1, 2, \ldots$ of the measurements and the range of possible errors of these measurements, and the computer itself produces both the value y and the error estimate for this value.

If we are planning to produce such an ideal solution, we need a software device, that would allow us to compute errors of arbitrary programs. This problem has two aspects: that of mathematics and that of software engineering. The mathematical aspect is to develop new improved methods of error estimates for different specific algorithms. The software engineering aspect is of two forms: we are planning to describe a device to compute these error estimates (some sort of a compiler) in software terms. Further, our goal is to provide precise numbers, with error estimates, to the user. Therefore this problem is an example of providing correct information, which is a problem in software engineering.

2. An illustrative example

In order to understand the models and the solutions better we'll give an illustrative example. This example is so simple that it does not constitute any problem from the computational viewpoint and is given here only for illustration purposes.

Suppose we cannot measure the voltage V directly, so we decided to measure it indirectly: i.e., measure the current I, the resistance R and then multiply the resulting values (i.e., apply Ohm's law). Suppose the result i of measuring current is 2.5 (lower case letters represent the actual measured quantities), the result r of measuring resistance is 4.0; then the resulting voltage value is $v = ir = 10.0$. The real value of voltage can be different from v, e.g., if we know that the precision of both measurements is 0.1 (the real values of I and R can differ from the measured ones by

no more than 0.1) then these real values can be, e.g., 2.59 and 4.08, in which case the real value V of voltage is $10.56\ldots$, i.e., the error in this case is $0.56\ldots$. In the other cases the error can be smaller or greater. It is impossible to enumerate all possible values of I and R (there are infinitely many), therefore the problem of estimating all possible values of error is sometimes non-trivial.

3. Mathematical formulation of the problem

To make our arguments easier to understand we'll start with a natural formulation of this problem. Then we'll show why this formulation is not very adequate to the user's problem and how to modify it.

3.1. Denotations and motivations for the mathematical formulation. By n we denote the number of all measurement results. The result of i^{th} measurement will be denoted by x_i; the real value of the measured quantity will be denoted by X_i. The difference between x_i and X_i will be denoted by d_i and called the error of the i^{th} measurement.

By $f(x_1, \ldots, x_n)$ we denote a value of the quantity y that corresponds to the values x_i of the measured quantities. The analytical expression for the function f may not be known, but there must exist a program that computes the value of f for any input data. So the real value Y of the desired quantity equals to $f(X_1, \ldots, X_n)$, the value y that is produced by the given program equals to $f(x_1, \ldots, x_n)$. The difference between them $d = y - Y$ is called an error of the resulting indirect measurements of y.

In the above example $n = 2, x_1 = i, x_2 = r, f(x, y) = xy$. X_1 is the real value I of current, $R = X_2$ is the real value of resistance, $v = ir$ is the result of our indirect measurement procedure and $V = IR$ is the real voltage.

3.2. Two kinds of errors. We are also supposed to know the estimates D_i for d_i, and from them we want to compute an estimate D for d. Following the traditional engineering approach (see, e.g., [1] [2]), we'll consider two kinds of errors: systematic and random errors.

Systematic errors. By saying that an error is systematic we mean (from the mathematical viewpoint) that the only thing we know about

the error d_i is that it belongs to a certain interval $[-D_i, D_i]$. In this case the possible values of X_i form an interval $[x_i - d_i, x_i + d_i]$. The resulting possible values of Y also fill the interval, and the estimate we are interested in is, in this case, the maximum possible value D of $|d| = |y - Y|$.

Random errors. Another typical situation in engineering applications is when the error is random, which means that d_i are random variables, and we know their probabilistic distributions. These distributions are usually considered to be independent; the average value of an error is 0; we know the mean square deviation D_i and assume that the probabilistic distribution is Gaussian. In this case the resulting error d is also a random variable, and we are interested in the mean square deviation D of this variable, i.e., $D_2 = E(d^2) = E((y - Y)^2) = E((f(x_1, \ldots, x_n) - f(X_1, \ldots, X_n)^2)$, where E means mathematical expectation.

3.3. First mathematical formulation. Now we are ready to formulate the first natural mathematical formulation of this problem.

The input data for our desired software device should include the following:

(1) an integer n; the number of measurement results
(2) a program f that, given n real numbers $x_1, \ldots, x_n$, computes the value y; the result of applying this program will be denoted by $f(x_1, \ldots, x_n)$
(3) n real numbers $x_i, : i = 1, 2, \ldots, n$; the measurement results
(4) n positive real numbers $D_i, : i = 1, 2, \ldots, n$; the estimates of the measurement errors
(5) a Boolean variable b; if $b = true$, the errors are systematic; if $b = false$, the errors are random.

In case $b = true$, we want to know the maximum possible value D of the expression $|d|$, where

$$d = f(x_1, \ldots, x_n) - f(X_1, \ldots, X_n) \tag{1}$$

and each X_i takes all possible values from the interval $[x_i - D_i, x_i + D_i]$.

In case $b = false$ we want to know the value $D = (E(d^2))^{\frac{1}{2}}$, where d is defined as above, $X_i = x_i + d_i$ and d_i are n independent normally

distributed random variables with average value 0 and standard deviation D_i.

The above formulation seems to comprise what the engineers really want. Its only drawback is that it is intractable or, using the precise mathematical term, NP-hard (Garey, Johnson, 1979). In other words, if we could solve this problem in reasonable computation time at least not exceeding some polynomial of the length of the input, then we would be able to solve all discrete problems in polynomial time, which is generally believed to be impossible.

It is intractable not because we permitted arbitrary programs for f. Rather the problem remains intractable even if we restrict ourselves to the case when the function f is a polynomial and $b = true$ (so the errors are systematic) ([4] [3]).

4. A new approach to ensure feasibility

In order to make the problem feasible let's take into consideration the fact that the value D that we are looking for is just the error estimate, so there is no need to compute it with great precision. A statement that "the precision of this measuring device is 10.3%" sounds crazy to an engineer; if we measure with precision about 10% (just 1 decimal digit), there's no sense to argue about the third decimal point in the error.

Engineers and physicists normally express this idea by saying that we can "neglect" terms quadratic in errors d_i. In computational terms this means that in order to compute any function F of d_i we expand the function in a Taylor series and neglect all quadratic (or higher order) terms, so that only linear terms remain.

In our case we are interested in the function d, that is described by equation 1. The values x_i are given, so the only thing that depends on d_i is $X_i = x_i - d_i$. Substituting this expression into equation 1, we get $d = f(x_1, \ldots, x_n) - f(x_1 - d_1, \ldots, x_n - d_n)$. Expanding this expression into a power (Taylor) series with respect to d_i and retaining only terms that either do not contain d_i at all or are linear in d_i, we conclude that

$$d = f_{,1}d_1 + \ldots + f_{,n}d_n \qquad (2)$$

where $f_{,i}$ means partial derivative of the function f with respect to the variable x_i and evaluated at the point $(x_1, \ldots, x_n)$.

How does this representation help to solve the computational problem of finding the value D?

Let's first suppose that the values of the derivatives $f_{,i}$ are already known. Then the expression 2 is just a linear combination of the values d_i with known coefficients. In this case both the maximization problem (in case of the systematic errors) and the problem of finding the mean square (in case of random errors) become easier to solve.

In case of systematic errors we know that d_i belongs to the interval $[-D_i, D_i]$. The term $f_{,i}d_i$ is an increasing function of d_i if $f_{,i} > 0$ and a decreasing function if $f_{,i} < 0$. Therefore if the derivative is > 0, then the maximum of this term is attained when d_i equals to the maximum possible value D_i, and the value of this term is then $f_{,i}D_i$. If the derivative is negative, then the maximum possible value of this term is attained when d_i takes the smallest possible value $-D_i$, and the correspondent value of this term is $-f_{,i}D_i$. Both cases are covered by one expression $|f_{,i}|D_i$.

The whole sum $d = f_{,1}d_1 + \ldots + f_{,n}d_n$ attains the maximum value if all its terms are maximal possible. Therefore the maximum D of this sum equals to the sum of the maxima of all these terms, i.e.,

$$D = |f_{,1}|D_1 + \ldots + |f_{,n}|D_n \tag{3}$$

Let's now consider the case of random errors. In this case d_i are independent random variables, with average $E(d_i) = 0$ and mean square $E(d_i^2) = (D_i)^2$. The fact that they are independent means, in particular, that $E(d_i d_j) = E(d_i)E(d_j) = 0$. Therefore the standard deviation D of the total error can be determined by applying the more or less standard statistical computations:

$$D^2 = E(d^2) = E((f_{,1}d_i + \ldots + f_{,n}d_n)^2)$$
$$= E((f_{,1})^2 d_1^2 + \ldots + (f_{,n})^2 d_n^2 + (f_{,1}f_{,2})d_{,1}d_{,2} + \ldots)$$
$$= E((f_{,1})^2 d_1^2) + \ldots + E((f_{,n})^2 d_n^2) = (f_{,1})^2 D_1^2 + \ldots + (f_{,n})^2 D_n^2$$

Therefore in this case

$$D = ((f_{,1})^2 D_1^2 + \ldots + (f_{,n})^2 D_n^2)^{\frac{1}{2}}. \tag{4}$$

Notice that in this case we do not need the supposition that the random errors d_i are normally distributed. This fact is very important, because in reality the probabilistic distribution of errors can be different from the Gaussian law.

The bottom line is that if we known the derivatives then it's easy to estimate D. So only one problem remains: How to compute the derivatives of f?

In case f is given by an analytical expression (as in the Ohm's law example), then by differentiating this expression we get an expression for the desired derivative, e.g, for Ohm's law $f_{,1} = x_2 = r$ and $f_{,2} = x_1 = i$. But what to do in the more difficult case, when f is given as a program and no mathematical expression for f is known? For example, when we analyze the geophysical data we apply different filtering procedures to the huge array of inputs. Of course this transformation (corresponding to some kind of Kalman filtering) can be viewed as a function, but no explicit expression for this function is known.

This problem may seem difficult to solve at first glance, but actually the same idea of "neglecting" second order terms helps here as well. Namely, when we assume that we can neglect the terms that are quadratic in errors, then from the engineering viewpoint we mean that not only the squares of the errors, but the squares of any numbers of the same order of magnitude are negligible. Therefore, if we take any n small real numbers $s_1, \ldots, s_n$ (small means of the same order of magnitude as possible errors d_i), then in all the expressions containing s_i we can neglect quadratic terms. In particular, just like for d_i, we conclude that

$$f(x_1 + s_1, \ldots, x_n + s_n) - f(x_1, \ldots, x_n) = f_{,1}s_1 + \ldots + f_{,n}s_n \qquad (5)$$

Therefore, substituting into the program f initial arrays $x_i + s_i$ for different s_i, we can compute linear combinations of the unknown derivatives, from which we can compute the derivatives themselves, e.g., in order to compute $f_{,1}$ we can use the values $s_1 = s$, $s_2 = \ldots = s_n = 0$ for some small s. Then the right-hand side of equation 5 becomes $f_{,1}s$, and we obtain the following expression for $f_{,1}$:

$$f_{,1} = (f(x_1 + s, x_2, \ldots, x_n) - f(x_1, x_2, \ldots, x_n))/s \qquad (6)$$

When s tends to 0, this formula turns into the definition of the partial derivative; actually this formula is used in numerical mathematics for computing partial derivatives.

Now we are ready to present the final mathematical formulation of our problem.

4.1. Final mathematical formulation. The input data for our desired software device should include the following:

(1) an integer n; the number of measurement results
(2) n real numbers x_i, : $i = 1, 2, \ldots, n$; the measurement results
(3) n positive real numbers D_i, : $i = 1, 2, \ldots, n$; the estimates of the measurement errors; we say that real numbers $s_1, \ldots, s_n$ are small if for every i the value $|s_i|$ does not exceed D_i;
(4) a program f that, given n numbers real numbers $x_1, \ldots, x_n$, returns some real result; the result of applying this program will be denoted by $f(x_1, \ldots, x_n)$
(5) a program that, given n real numbers $x_1, \ldots, x_n$ and n small real numbers $s_1, \ldots, s_n$, returns the value $f(x_1, \ldots, x_n) + f_{,1}s_1 + \ldots + f_{,n}s_n$, where $f_{,i}$ means the value of i^{th} partial derivative of the function f (i.e., the function computed by the program f) at the point $(x_1, \ldots, x_n)$;
(6) a Boolean variable b; if $b = true$, the errors are systematic; if $b = false$, that the errors are random.

In view of the above arguments the second program can be implemented simply by using $x_i + s_i$ as an input of the first (main) program.

In the case $b = true$, we want to know the value given by equation 3; in the case $b = false$, we want the value given by equation 4.

Our formulation is already final and consistent, but does represent a mixture of pure mathematics and engineering assumptions. This approach may be validated using the techniques of nonstandard numbers [6], but this approach provides only a non-algorithmic justification and we need a computer algorithm, which we derive below.

5. Algorithms that estimate errors

5.1. Estimation using partial derivatives. This idea is very simple:

we have explicit formulas allowing us to compute D in case we know the partial derivatives, and we have a method to compute these derivatives, so we compute them, substitute the result into equation 3 or equation 4 and get the desired answer. Since it is already a working tool, let's describe the algorithm step by step:

(1) We assume that the value $y = f(x_1, \ldots, x_n)$ has already been computed, in other words, the data processing algorithm f has already been applied to the input data x_i.

(2) Computing partial derivatives. For every i from 1 to n do the following:

(a) compute the vector $(s_1, \ldots, s_n)$ with coordinates $s_1 = s_2 = \ldots = s_{i-1} = s_{i+1} = \ldots = s_n = 0$ and $s_i = D_i$;

(b) substitute $x_1 + s_1, \ldots, x_n + s_n$ into f, thus computing $y_i = f(x_1 + s_1, \ldots, x_n + s_n)$

(c) compute $f_{,i} = (y_i - y)/s_i$

(1) (3) compute D from D_i and $f_{,i}$, using equation 3 or equation 4, as appropriate

If n is small (like $n = 2$ in the Ohm law example) this algorithm works fine. But in some cases we need to process a lot of data in order to get the desired value y. In this case n is big, and the data processing program f takes a lot of computer time. The above-described algorithm calls the function f n times (once for each of n partial derivatives), therefore its running time is n times greater than the already (possibly) large running time of f itself (and n is also big!).

For the case of random errors there is a good way out: computer simulation of these errors, or the Monte-Carlo method. The idea is that we use the computer's (pseudo-)random number generator to generate s_i which are distributed according to the Gaussian law with 0 average and standard deviation D_i. (Actually the pseudo-random generators return a value that is normally distributed with standard deviation, so we have to multiply this value by D_i to get the desired s_i). Then we substitute $x_i + s_i$ into f, and compute the difference between the resulting value and the value $y = f(x_1, \ldots, x_n)$, that was obtained from the initial data. This difference $d = f(x_1 + s_1, \ldots, x_n + s_n) - f(x_1, \ldots, x_n) = f_{,1}s_1 + \ldots + f_{,n}s_n$ is a linear combination of the normally distributed independent random variables, and therefore its probabilistic distribution is also Gaussian.

with standard deviation $D = ((f_{,1})^2 D_1^2 + \ldots + (f_{,n})^2 D_n^2)^{\frac{1}{2}}$ equal to the desired value. So if we repeat this experiment several (N) times, then the resulting values $d_1, \ldots, d_N$ are distributed according to this Gaussian law and therefore we can use the standard statistical formula to determine D:

$$D = (((d_1)^2 + \ldots + (d_N)^2))/(N-1))^{\frac{1}{2}} \tag{7}$$

The precision of this formula is determined by the size N of the sample (and thus does not depend on n), e.g., if we want to estimate D with relative precision 20% (that is quite sufficient from an engineering viewpoint, since 1.2% precision is actually the same as 1%) we need $N = 25$. Therefore we need only 25 calls of f; for large n this is smaller than the n calls of the first algorithm. So we arrive at the following Monte-Carlo algorithm for random errors:

For $k = 1, 2, \ldots, N = 25$ we repeat the following:

(1) use the random number generator to generate n numbers r_i that are normally distributed with 0 average and standard deviation 01

(2) substitute $x_i + D_i r_i$ into f and compute $d_k = f(x_1 + D_1 r_1, \ldots, x_n + D_n r_n) - f(x_1, \ldots, x_n)$

(3) compute $D = (((d_1)^2 + \ldots + (d_N)^2))/(N-1))^{\frac{1}{2}}$

That we can use the Monte-Carlo approach to estimate random errors is not very surprising. It's interesting that we can use it for systematic errors as well! (For the detailed exposition and practical application examples see [5]).

This method is based on the following property of a special distribution law called Cauchy law, with a probabilistic density $p(z) = const\, z/(z^2 + p^2)$, where p is called the scale parameter of this distribution. This property is that if $z_1, \ldots, z_n$ are independent random variables, and each of z_i is distributed according to the Cauchy law with parameter p_i, then their linear combination $z = c_1 z_1 + \ldots + c_n z_n$ also distributes according to a Cauchy law, with a scale parameter $p = |c_1| p_1 + \ldots + |c_n| p_n$. Therefore, if we take s_i that are Cauchy distributed with parameters D_i, then the value $d = f(x_1 + s_1, \ldots, x_n + s_n) - f(x_1, \ldots, x_n) = f_{,1} s_1 + \ldots + f_{,n} s_n$ is Cauchy distributed with the desired parameter $D = |f_{,1}| D_1 + \ldots + |f_{,n}| D_n$. So repeating this experiment N times, we get N values $d_1, \ldots, d_N$ that are

Cauchy distributed with the unknown parameter, and from them we can estimate D.

There are two questions to be solved:

(1) how to simulate the Cauchy distribution
(2) how to estimate the parameter D of this distribution from a finite sample

Simulation can be based on the functional transformation of uniformly distributed sample values: $s_i = D_i \tan(\pi(r_i - 0.5))$, where r_i is uniformly distributed on the interval $[0, 1]$.

In order to estimate D we can apply the Maximum Likelihood Method $p(d^1)p(d^2)\ldots p(d^N) \to max$, where $p(z)$ is a Cauchy distribution with the unknown D. When we substitute the above-given formula for $p(z)$ and equate the derivative of the product with respect to D to 0 (since it is a maximum), we get an equation

$$(1 + (d^1/D)^2)^{-1} + \ldots + (1 + (d^N/D)^2)^{-1} = N/2 \qquad (8)$$

The left-hand side is an increasing function that is equal to $0 (< N/2)$ for $D = 0$ and $> N/2$ for $D = \max(d^k)$, therefore its solution can be found by applying a bisection method to the interval $[0, max(d^k)]$.

The precision of this estimate again depends only on N (20% precision is achieved for $N = 50$), so this method uses $N = 50$ calls of f, and in the case $n \gg 50$ it saves a lot of computation time.

Thus the Monte-Carlo algorithm for systematic errors:

For $k = 1, 2, \ldots, N = 50$ repeat the following:

(1) use the random number generator to compute n numbers r_i, : $i = 1, 2, \ldots, n$, that are uniformly distributed on the interval $[0, 1]$;
(2) compute $s_i = D_i \tan(\pi(r_i - 0.5))$
(3) substitute $x_i + s_i$ into the function f and compute $d_k = f(x_1 + s_1, \ldots, x_n + s_n) - f(x_1, \ldots, x_n)$
(4) compute D by applying the bisection method to solve the equation $(1 + (d^1/D)^2) - 1 + \ldots + (1 + (d^N/D)^2)^{-1} = N/2$

If we have f as a program say in Pascal it's easy to implement all these three methods as Pascal programs that call f as a subroutine. We have

actually implemented these algorithms into two working tools that create the error estimation programs from the programs in Pascal and Fortran.

6. The use of parallel processing

6.1. Why parallelization is possible and helpful here. All these error estimating algorithms consist of running the given data processing program f with simulated data $x_i + s_i$, and then in transforming the results d_k of applying f into the desired estimate D (e.g., in the first algorithm this simulation helped to compute the partial derivatives of f, from which we computed D).

Computations that lead to s_i, and the computations that get D from d_k are rather easy, the most time-consuming part of these algorithms is calling the program f for various simulated data. Therefore the natural time-saving idea is to make all these calls in parallel. In case we have sufficiently many parallel processors (n for the first algorithm, 25 and 50 for the 2^{nd} and the 3^{rd}), then all these calls can be made in parallel, so we can reduce the computation time of the error estimate practically to the computation time of the data processing itself, i.e. in addition to the processed data get its error estimate practically in no additional time (at the expense of using additional processors).

In case we do not have that many processors, we can still distribute the jobs between the available processors, thus diminishing the total computation time from T to approximately T/P, where P is the number of the processors, e.g., if we use the first algorithm for a program with $n = 20$ input data, and 4 processors are available, then we can make the first processor compute the partial derivatives with respect to the variables x_{1-5}, the second for x_{6-10}, the third for x_{11-15} and the fourth for x_{16-20}, thus diminishing the total running time for error estimate from $20T_f$ (where T_f is a running time of f itself) to $5T_f$. In the Monte-Carlo methods we can distribute 25 (or 50) repetitions between those processors, with the same time-saving effect.

Theoretically the more processors we have the quicker are the results. But in real parallel systems a lot of time is consumed on communication protocols, information exchange, waiting in the queues, etc. The more processors we have, the more time-consuming all these communication procedures become, and they seriously impact the whole computation

process. So if we implement our parallelized algorithms on real parallel systems with many processor, this additional time will add to our running time and thus worsen our theoretical estimates.

The "overhead" of additional time wasting is difficult to avoid if we really want several processes to send each other a lot of information during the computational process. But in our case during the main ("calling f") stage no communication between the processors is necessary, and the only communication we need is:

(1) sending the values x_i and D_i to the processors before that main stage, and
(2) sending the resulting values d_k from the processors to one processor which will compute D from these values

There are no communication conflicts, so we don't want to waste time on protocols.

The fact that we need to send the same values to all the processors means that we need small shared memory. The second communication task can be also solved by a small shared memory if we allocate 1 word for every processor (at the expense of maximally 50 additional words). In this case there can be no attempts to write contradictory data into this common memory, so we can implement it in the easiest possible way, e.g., we can choose one of the processors (that actually reads x_i) to be the Coordinator. In the memory of each processor we select two small pieces: for drop and for pick. In the beginning the signal is picked from the Coordinator and dropped consequently into the "drop" part of all other processors. At the end the same consequent process can be used to "pick" the results from the processors and drop them into the Coordinator's memory. So what we need is to connect all the processors by a loop and organize a continuous pick-drop process: a signal is picked from the pick locations and dropped into the Coordinator's memory.

Such a simple loop system has been produced by Septor Electronics for use in machinery control applications [7] [8].

We have implemented the above-described algorithms on a system based upon the Septor Electronics boards and indeed achieved the desired speedup: P times if we use P processors. Figure 1 displays sample results.

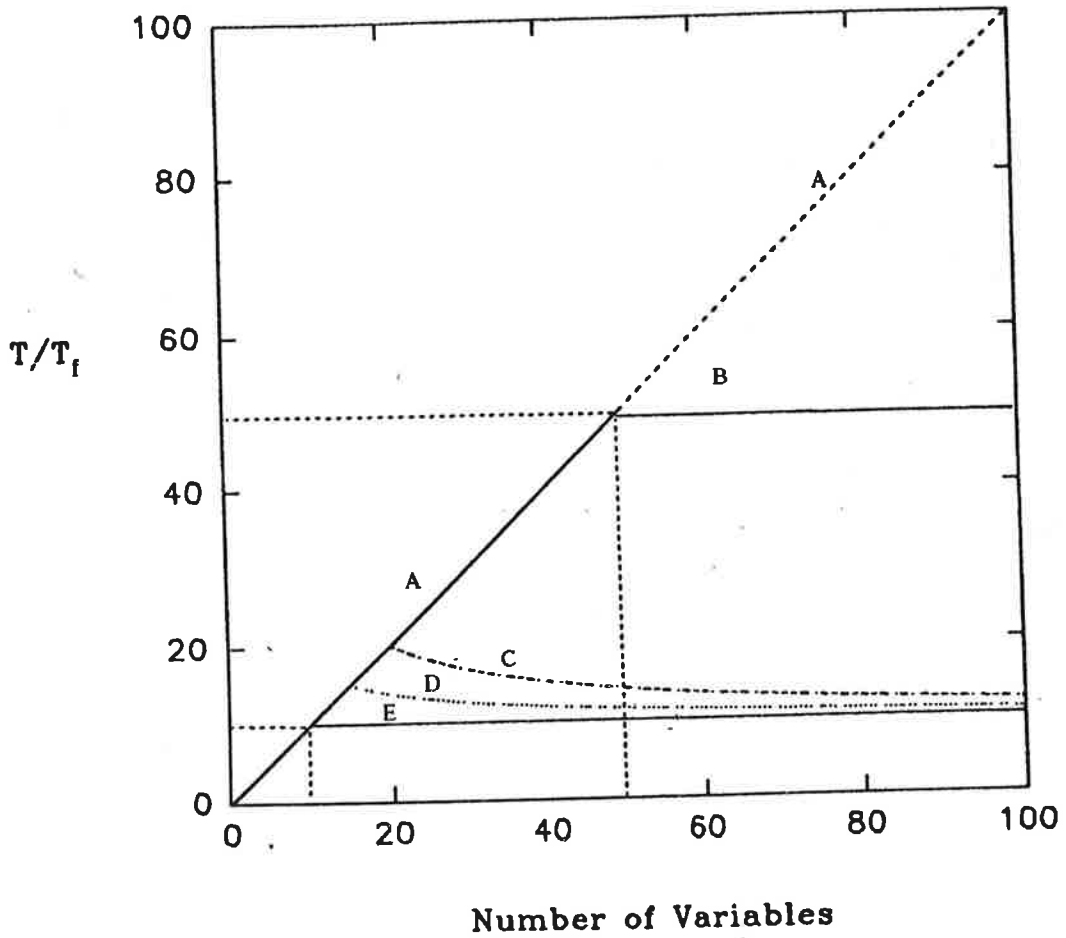

Figure 1: Efficiency on Parallel Processors. Notation is: T, execution time for error estimates; T_f, time to compute a function; n, number of variables; A, estimation using partial derivatives; B, sequential Monte Carlo method; C–E, 5 processor Monte Carlo method with C and D for differing algorithmic complexity and E the theoretical limit.

Acknowledgements. This work has been supported by a grant from The Institute for Manufacturing and Materials Management and by an equipment donation from Septor Electronics.

References

1. Clifford, A. A. *Multivariate Error Analysis*, J. Wiley & Sons, New York, 1973.

2. Fuller, W. A. *Measurement Error Models*, J. Wiley & Sons, New York, 1987.

3. Gaganov, A. A. *Computational complexity of the range of a polynomial in several variables*, Cybernetics, 418–421, 1985. (In Russian; this journal is translated into English.)

4. Garey, M., and Johnson, D. *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

5. Kreinovich, V. Ya., and Pavlovich, M. I. *Error Estimate of the Result of Indirect Measurements by Using a Calculational Experiment*, Measurement Techniques, **28**, 201–205, 1985.

6. Robinson, A. *Nonstandard Analysis*, North-Holland, Amsterdam, 1966.

7. Roberts, R. *AI Enhanced Transfer Lines*, Programmable Controls, May, 105–108, 1989.

8. Hardin, J., and Taylor, J. *A Distributed Parallel Processing System Can Solve Today's Complex Automated Machine Control Problems*, Proc. 19^{th} Annual International Programmable Controllers Conference, 1990.

Computer Science Department
The University of Texas at El Paso
El Paso, TX 79968, USA