U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 5417

Computer Systems Laboratory

A Simple Scalability Test for MIMD Code

Gordon Lyon
Raghu Kacker

U. S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Gaithersburg, MD 20899

June 1994

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

Partially sponsored by the
Advanced Research Projects Agency

A Simple Scalability Test for MIMD Code

Gordon Lyon, Advanced Systems Division
Raghu Kacker, Statistical Engineering Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Gaithersburg, MD 20899

Partially sponsored by the Advanced Research Projects Agency.

June 1994

TABLE OF CONTENTS

Page

A Simple Scalability Test for MIMD Code

Gordon Lyon, Advanced Systems Division
Raghu Kacker, Statistical Engineering Division

Code scalability, crucial on any parallel system, determines how well parallel code avoids becoming a bottleneck as its host computer is made larger. Scalability of computer code can be estimated by statistically designed experiments that empirically approximate a multivariate Taylor expansion of the code's execution response function. Each suspected code bottleneck corresponds to a first-order term in the expansion, the coefficient for that term indicating how sensitive execution is to changes in the suspect location. However, it is the coefficients for second-order interactions between code segments and the number of processors that are fundamental in discovering which program elements limit parallel speedup. Extending an earlier formulation, a new unified view via these second-order terms yields an informal scaling test of high utility in code development.

Key words: designed experiments; MIMD scalability; parallel processing; performance evaluation; sensitivity analysis.

NIST Task 40131 and ARPA Task 7066 provide major support for this work. Contributions of NIST are not subject to U.S. copyright. To appear in the Proceedings, IEEE Second International Symposium on Software Metrics, October 24-26, 1994, London, England.

1. Background

From earliest days, computing has employed various performance statistics to guide the design, writing and tuning of code [7]. By far the most popular statistics are counts and timings that profile computing demands of pieces of code within a program. These pieces can be procedures, segments of straight-line code or even single statements [3][5]. On a uniprocessor--the classical von Neumann machine--gathered profiles reveal clearly where a program consumes available computing cycles.

Profiles become more confused on multiple-instruction, multiple-data (*MIMD*) stream parallel systems. For one thing, the straightforward linear thread of serial computation is replaced by many concurrent threads. A consequence of this is a computation state space that grows combinatorially with the size of the host. Whereas a single executing thread clearly defines a program state, this is now only part of a product state space defined over all processors. To see the problem, imagine a serial program with six major states. Suppose a parallel version of this code is replicated on each processor node of a MIMD system. This parallel program running on 16 processors now has a (naive) state space of $6^{16} \approx 2.8 \times 10^{12}$ states! Now if the system is symmetric in its nodes, it may not matter whether node 5 is substate 3 and node 6 is substate 4 or *vice versa.* This will merge states that are only different orderings of the same (macro)state. Nonetheless, as the *scalable host system* grows to 256 or more nodes, state-based execution categories become awkward and problematic.

Yet another profiling problem arises with the parallel version; some substates are dependent upon actions in neighboring nodes. Since parallel copies of the program are cooperating together, this is not unexpected. A substate may wait for a message. However, profiling information on these latencies (waiting periods) is not always a reliable hint on the true causes of the waitings. In many cases, latencies are like slack variables. They indicate waste, but fail to supply a true indication of problem source; *e.g.*, which node was too slow in sending which message, and at what point? In contrast, the serial program's states often identify a certain amount of computing that must be done. A high profile indication for a state then indicates (i) a heavy computation load in the state, or (ii) poor quality code. A parallel substate with a latency, which is a dependency, lacks such a clear indication.

Thus, applied to scalable parallel programs, conventional execution profiles and their interpretation schemes are often stressed badly. Nonetheless, successful code improvement depends upon identifying the more sensitive segments within a program, so that severe bottlenecks can be corrected. It is fortunate that execution profilings are not the only avenue to performance improvement.

1.1 Correlating Settings and Responses

Another approach to code tuning employs the *statistically designed experiment (DEX)*, which handles program and system together as an amorphous entity that has controllable input settings to *factors* [6]. Factors are code segments suspected of being bottlenecks; these factors are set (or tested) via benign changes to their code. System size is another factor; it is determined by the number of processors assigned to a program. A statistically designed experiment then correlates runs of the variously-treated program/system entity (each run setup is a *treatment*) against a response measured for each run (*e.g.*, overall program run time). This powerful *macromodeling* or *curve fitting* perspective predicts how factor settings and interactions among settings influence performance [1][2][4]. For instance, designed experiments are very good at characterizing factor interactions, which as mentioned are not unusual among cooperating processes. *State* in the DEX approach is shifted from the executing program, where the state space is huge and time is an element, to a setting in source code. Smaller and static, this input-defined space draws upon a large body of DEX methods to simplify its handling and interpretation [1][2]. Appendix A provides a glimpse of the breadth of DEX methods. DEX is highly adaptable to parallel computing investigations, for its approach is flexible and scalable.

2. Parallel Code Scalability

Code scalability, always crucial on any parallel system, determines how well a section of code avoids becoming a bottleneck as its host computer is made larger. A recent note by Snelick, *et al.*, presents an interesting method of determining the scalability prospects of parallel code [8]. Their approach employs DEX and *synthetic perturbation (SP)* [6], the latter arising because applying DEX to test software performance requires economical ways of modifying code efficiencies. Naturally occurring, adjustable parameters are unlikely in arbitrarily chosen code segments. Segments lacking natural parameters can be recoded, but these recodings must be checked for correctness, an impracticality when the set of segments is large. Synthetic perturbation, SP, is an alternate method of code modification that employs artificial means [6]. The attractive SP *simulates* local, easily adjusted efficiency changes, which in turn drive DEX sensitivity analyses. Each SP, typically extra code that causes delay, can be inserted or removed smoothly without interfering with actual computations. The new test uses the high setup efficiencies inherent in SP.

DEX yields an approximate multivariate Taylor expansion of the overall program/system execution response. Details of this correspondence between DEX and the Taylor expansion are shown in Appendix B. Each suspected code bottleneck in the code will correspond to a first-order term in the expansion, the coefficient for that term (called a *main effect*) indicating how sensitive execution is to changes in the suspect factor. However, it is the coefficients for second-order interactions between code segments and the number of processors that express which segment improvements best promote parallel speedup. A new, unified view of the second-order terms yields a simple scaling test. The test has high utility in focusing parallel code improvement efforts.

2.1 Simplifying Earlier Results

Snelick, *et al.*, base their scalability test for parallel code upon orderings of main effects [8]. In their approach, system scale, s (the number of processors), is handled differently from other factors. There is a separate preliminary stage for each of two distinct settings of s. A second stage then compares the two DEX-developed first stage sensitivities. This comparison is not as simple as it might be, involving as it does orderings in two distinct tables. Promoting s into regular factor status yields one common view of the sensitivities: Interpretation is straightforward.

2.1.1 A Scaling Test. This example from Snelick, *et al.*, involves comparing sensitivities of code segments from a 3500-line image understanding benchmark (*IUB*) against the size of the host system [8]. Tables 1 and 2 show rankings of the *IUB* software sensitivities for $s=8$ and $s=24$ processors, respectively. Main effects (what mathematical statisticians call *half effects* [2]) in the third columns are sensitivities--slope coefficients--of *IUB*'s response to changes in individual factors. The benchmark is most sensitive to changes in factors with larger effects. By comparing rank-orderings of effects in the two tables, estimates can be made about scalabilities of various segments of code. As a factor gains relative importance, it is increasingly worth inspecting. For instance, factors $F2, F1$, and $F4$ migrate to higher rankings in Table 2, drawing attention to their parent routine, *Connected Components*.

2.1.2 Test with Combined Tables. The comparison between ordered effects of Tables 1 and 2 does not fully exploit available information. To see this, imagine a new midpoint Table 3 for s = (8+24)/2 = 16 processors. For Table 3 effects, *linearly* interpolate main effects from Table 1, column A and Table 2, column B effect-by-effect as $(A+B)/2$. Similarly, incorporate second-order interaction effects $(A-B)/2$ that predict *changes* in main effects *vis-`a-vis* scaling factor s. Linear interpolation is compatible with the (linear) response expansion in Tables 1 and 2. (Assumptions behind the expansion are discussed shortly.) The interaction effects $\beta_{i,s}$ in Table 3 predict scaling success *explicitly* and *quantitatively*. For instance, the scaling interaction $\beta_{F2,s} = 0.09$ for factor $F2$ is greater than zero; factor $F2$ therefore does not scale, since its main effect grows larger as processors are added. A more negative interaction $\beta_{i,s}$ in Table 3 implies a more scalable factor.

Rank	Factor (i)	Main Effect (β_i) §	Routine	Construct
		A		
1	F17	6.03	Grad Magn	*for*
2	F26	5.46	Med Filt	*while*
3	•F2	5.26	Conn Comp	*for*
4	•F1	3.94	Conn Comp	*function*
5	•F4	3.84	Conn Comp	*while*
6	F25	2.01	Med Filt	*for*
7	F20	1.67	Match	*function*
8	F29	1.36	Probe	*for*

Source: *R. Snelick*

§ *Standard Error (Uncertainty): ±0.04*

Table 1: IUB Main Effects with Eight Processors.

Rank	Factor (i)	Main Effect (β_i) §	Routine	Construct
		B		
1	**•F2**	**5.43**	**Conn Comp**	*for*
2	**•F1**	**3.95**	**Conn Comp**	*function*
3	**•F4**	**3.93**	**Conn Comp**	*while*
4	**F17**	**2.14**	**Grad Magn**	*for*
5	**F26**	**2.03**	**Med Filt**	*while*
6	**F20**	**1.55**	**Match**	*function*
7	**F29**	**0.95**	**Probe**	*for*
8	**F25**	**0.75**	**Med Filt**	*for*

Source: *R. Snelick*

§ *Standard Error (Uncertainty): ±0.12*

Table 2: IUB Main Effects with 24 Processors.

Factor (i)	Main Effect (β_i) §	Interaction Effect ($\beta_{i,s}$) §	Scaling Implication
	(A+B)/2	(A-B)/2	
F2	5.35	+0.09	*not scaling at all*
F17	4.09	-1.95	*good--dropping fast*
F1	3.95	+0.01	*poor (not scaling)*
F4	3.89	+0.05	*poor*
F26	3.75	-1.72	*another good factor*
F20	1.61	-0.06	*poor*
F25	1.37	-0.64	*will drop below F29*
F29	1.16	-0.21	*? not clear*

Source: Tables 1 & 2

§ *Standard Error: ±0.07*
Estimations for s=16 processors.

Table 3: Scalabilities Estimated via Second-Order Terms

3. Scalability Assay for an Isolated Factor

While Table 3 is a definite improvement over Tables 1 and 2, it still tests numerous code segments together. In the next refinement, the scaling test is simplified to work on a single code segment.

3.1 An Example that Generalizes

Imagine a scaling assessment for a routine *cd* (), which is a major parallel phase of a hypothetical parallel program. By obvious substitution of subject code or measured response, steps and formulae for this exercise generalize to other specimen programs and systems. The evaluation employs DEX to establish how changes in *cd* () and s affect system performance (here with overall program run time as the measured response).

Factor *cd* () is represented with setting $X_{cd} = -1$ for the original *cd* () and $X_{cd} = +1$ for another modified copy *cd′* () with an added delay. These variants offer performance differences needed for the investigation. As mentioned earlier, the SP delay simulates efficiency changes in *cd* (). This delay is much easier to insert and adjust than are actual modifications to the computational code. The added SP must be large enough to have an effect above background experiment noise, but not be so large as to slow or distort experiments significantly (*cf*. discussion in [6]). $X_s = -1$ denotes a host system configuration of p processors, while $X_s = +1$ indicates a larger configuration of $p' > p$ processors. Notice that settings for code (*original/delayed*) and host scale (*smaller/larger*) both correspond to input domains of $[-1,+1]$. Statistical experiment designs and analyses are always expressed via such normalized input and must be interpreted within this context. Continuing with input settings, the designed experiment (DEX) is a collection of *trials*. Each trial designates a distinct combination of settings (treatment) for X_{cd} and X_s for which a *response* R is *measured* and *recorded*. *Replications* are repeated trials. Two settings for each of *cd* () and *scale* generate $2^2 = 4$ distinct experiment trials.

3.1.1 A Short Scaling Test for cd(). Each row of Table 4 records treatments and their corresponding *measured* system responses (program run times), which are labeled $a-d$.

Xcd	Xs	R, seconds	
-1	-1	40	(a
+1	-1	44	(b
-1	+1	24	(c
+1	+1	29	(d

Table 4. Scaling Experiment

The above data are now applied to a *response surface model* that is linear in coefficients of terms X_{cd}, X_s and $I_{cd,s}$. This linear model is adequate. Since code is being changed constantly during improvement, highly accurate predictions may actually waste effort. However, a nonlinear response model can be employed (see [1]). This might, for example, capture the hyperbola-like curvature in the response that would be expected from scaling with ideal load balancing and low levels of inter-process communication. As it is, the linear (coefficients) model includes an important interaction term $I_{cd,s} = X_{cd} * X_s$. Whenever X_{cd} and X_s agree in sign, $I_{cd,s} = +1$. Otherwise $I_{cd,s} = -1$. Thus, the response model is:

$$\beta_{cd}X_{cd} + \beta_s X_s + \beta_{cd,s}I_{cd,s} + \mu = R(X_{cd}, X_s) \qquad (i)$$

In Table 4, X_{cd} and X_s (and by extension, $I_{cd,s}$) are each individually fixed at -1 or $+1$ for each of four trials. Responses R are then measured (labels $a-d$). This leaves four unknowns--three effect coefficients $\{\beta\}$ and a constant term, μ. Inserting values for $X_{cd}, X_s, I_{cd,s}$ and R into equation (i) yields four simple equations:

$$-\beta_{cd} - \beta_s + \beta_{cd,s} + \mu = 40$$

$$+\beta_{cd} - \beta_s - \beta_{cd,s} + \mu = 44$$

$$-\beta_{cd} + \beta_s - \beta_{cd,s} + \mu = 24$$

$$+\beta_{cd} + \beta_s + \beta_{cd,s} + \mu = 29$$

Because the experiment design is orthogonal, these equations are solved easily [2]. The pattern of signs for each unknown in the above equations is crucial. For example, unknown μ is the mean response over all trials and is obtained by averaging the column R of observed responses. Note that in all four equations above, μ is added. Using this pattern:

$$\mu = \frac{+a + b + c + d}{4} = \frac{40 + 44 + 24 + 29}{4} = 34.25 \qquad (r1)$$

Other coefficients similarly employ columns of values from Table 4. For β_{cd}, multiply the column X_{cd} term-by-term with column R and average:

$$\beta_{cd} = \frac{-a + b - c + d}{4} = \frac{-40 + 44 - 24 + 29}{4} = 2.25 \qquad (r2)$$

β_{cd} is a *slope*, the rate of change of $\bar{R}$ as X_{cd} varies, $\frac{\Delta \bar{R}}{\Delta X_{cd}}$. Imagine the average response when X_{cd} is set +1 minus the average response when X_{cd} is set −1; the net result is then divided by two because the input domain is $1-(-1)$, a distance of two. Rewriting $r2$ shows this:

$$\beta_{cd} = \frac{\left[\frac{44+29}{2}\right] - \left[\frac{40+24}{2}\right]}{2}$$

Coefficient β_s is the overall change in response per additional processor. Following the computation template with column settings for X_s and values for R yields:

$$\beta_s = \frac{-a - b + c + d}{4} = \frac{-40 - 44 + 24 + 29}{4} = -7.75 \qquad (r3)$$

Discovery and evaluation of the coefficient $\beta_{cd,s}$ for interaction term $I_{cd,s}$ is *the keystone to the scalability test.* $\beta_{cd,s}$ expresses the sensitivity of a sensitivity; it is the difference per unit X_s of the difference in R per unit X_{cd}:

$$\beta_{cd,s} = \frac{\Delta^2 R}{\Delta X_{cd} \Delta X_s} = \frac{\left[\frac{d-c}{2}\right] - \left[\frac{b-a}{2}\right]}{2}$$

The simplified formulation again uses Table 4. Multiply column X_{cd} term-by-term with X_s to derive $I_{cd,s}$, then multiply $I_{cd,s}$ term-by-term with R:

$$\beta_{cd,s} = \frac{+a-b-c+d}{4} = \frac{+40-44-24+29}{4} = 0.25 \qquad (r4)$$

With $r1$–$r4$, the four unknowns have been solved. The solution is centered at point $(X_{cd}, X_s) = (0, 0)$, the average input setting over all trials.

Interpretation of the results depends upon further information on experimental error. *Standard error*, SE, is a measure of experiment noise (uncertainty). Noise-generated effects (worthless coefficients) are samples from a normal distribution centered at zero with standard deviation estimated by SE [2]. Any coefficient (effect) can be real or it can be from noise. By setting the noise range as $0 \pm 2SE$, an approximate 95% confidence interval is established (assuming that errors are normally distributed).

If SE_β is known for effects from previous experience, it can be used. (The example for *cd*() assumes this circumstance.) Otherwise replications of trials should be run. Let these duplicates be a', b', c' and d'. The standard deviation of effect (or coefficient) noise is then estimated as:

$$SE_\beta = \frac{1}{4}\left[(a-a')^2 + (b-b')^2 + (c-c')^2 + (d-d')^2\right]^{1/2}$$

Standard error for the mean performance, μ, is $SE_\mu = SE_\beta / 2$ (see reference [2]). Effects are calculated as before, only substitute response averages, *e.g.*, $\bar{a} = (a + a')/2$, in formulae $r1$ through $r4$ where now single measurements a–d appear.

3.1.2 Example Interpreted. The $cd()$ terms from $r1$–$r4$ (above) are:

$$\begin{aligned} \mu &= 34.25 \\ \beta_s &= -7.75 \\ \beta_{cd} &= 2.25 \\ \beta_{cd,s} &= 0.25 \end{aligned}$$

$$\begin{aligned} SE_\beta &= \pm 0.10 \text{ (from prior experience)} \\ SE_\mu &= \pm 0.05 \end{aligned}$$

All three effects (the β terms) are significant at an approximate 95% confidence interval of $2SE_\beta = 0 \pm 0.20$. This interval is a noise band around zero. Quantity $\mu = 34.25$, the average of response R across all four trials, establishes a context for evaluating magnitudes of the sensitivities, $\{\beta_i\}$. Scaling sensitivity of the *overall system* is expressed by $\beta_s = -7.75$. Had $\beta_s \geq -0.20 = -2|SE|$, there would be little point in adding further processors; at best, they would not slow computation. In this example, good scaling yields are still available. Next, consider term $\beta_{cd} = 2.25$, which expresses the sensitivity of the system response to changes in $cd()$. Since β_{cd} is well outside the $2SE_\beta$ noise band, efficiency changes to cd from delays are significant. But the main question is whether β_{cd} grows or diminishes with s, the scaling factor. The interaction term, $\beta_{cd,s} = 0.25 > 0.20 = 2|SE_\beta|$, indicates a failure in scaling for β_{cd}, the latter term growing mildly with increasing scale s. R is thus increasingly sensitive to changes in $cd()$ as processors are added.

3.1.3 Remaining Circumstances. Whenever both $\beta_s < -2|SE_\beta|$ and $\beta_{cd,s} < -2|SE_\beta|$, the scalability assessment of $cd()$ is not immediately obvious through inspection. A *scaling proportionality* is one useful criterion that can be applied. For the criterion to make sense, all terms μ, β_s, β_{cd}, and $\beta_{cd,s}$ must be significant.

The idea is to restrict the estimated sensitivity of $cd()$ under scaling to at most its current relative importance β_{cd}/μ. Thus, if through speedup μ becomes half of its present value, then the corresponding β_{cd} must be *at most* half of its current value. The scaled value of β_{cd} is predicted via slope $\beta_{cd,s}$, so this slope must descend at least in proportion to β_s, which predicts scaling change in μ. Expressing this in terms of the known values:

$$\beta_{cd,s} \leq \left(\frac{\beta_{cd}}{\mu} \right) \beta_s < 0 \implies cd() \textit{ scales.}$$

4. Conclusions

Strong, general mathematical formulations from DEX serve excellently in designing a scalability test for parallel code. Inserted synthetic delays further rid any need to modify original coding; this expedites DEX setups and avoids recoding errors. Methods and calculations for the scalability test apply to any segment of parallel code. The test requires neither special instrumentation nor custom, detailed models. Independent of system, programming language, and software specimen, the scalability test provides a portable, quick and easy assay that can guide deeper investigations into causes.

Acknowledgment. Thanks to R. Snelick for sharing additional data on the Image Understanding Benchmark and to T. Wheatley for helpful comments.

5. References

[1] T.B. Barker. *Quality by Experimental Design,* (Marcel Dekker, Inc., New York, 1985).

[2] B.E.P. Box, W.G. Hunter and J.S. Hunter. *Statistics for Experimenters*, (John Wiley & Sons, New York, 1978).

[3] S. Graham, P. Kessler, and M. McKusick, Gprof: A call graph execution profiler, *Proc., ACM SIGPLAN Symp. on Compiler Construction*, June, 1982.

[4] R. Jain. *The Art of Computer Systems Performance Analysis,* (J. Wiley & Sons, New York, 1991).

[5] D. Knuth, An empirical study of FORTRAN programs, *Software--Practice and Experience* 1, (1971) 105-133.

[6] G. Lyon, R. Snelick, and R. Kacker, Synthetic-perturbation tuning of MIMD programs, *The Journal of Supercomputing* 8, (1994) 5-27.

[7] J. von Neumann and H.H. Goldstine, *Planning and Coding of Problems for an Electronic Computing Instrument*, (Institute for Advanced Study, Princeton, N.J. [3 vols.], 1947-1948). Reprinted in von Neumann's *Collected Works* (A. Taub, ed.), Vol. 5, (Pergamon Press, Oxford, 1963).

[8] R. Snelick, J. Ja'Ja', R. Kacker, and G. Lyon, Synthetic perturbation techniques for screening shared-memory programs, *Software--Practice and Experience*, (to appear).

6. Design of Experiments (Appendix A)

Statistically designed experiments (DEX) are strategies to maximize learning at minimum cost. A typical DEX involves a number of input variables called factors and one or more output variables called responses. The possible settings of the input variables delineate a multi-dimensional experimental region. This region is usually quite large. In its simplest form, a DEX specifies the test points in the experimental region at which the response is evaluated. Combinatorial mathematics is used in determining the test points, thereby ensuring a geometrically balanced coverage of the experimental region. A map of the response, thus obtained, provides an estimate of relationships between factors and responses. Certain tactics, called randomization, blocking, and replication, are used in DEX to enhance the validity of the response map. Although DEX are usually used in physical science experimentation, their use in computer experiments is gaining momentum. The key difference is that, in computer experiments, the responses arise from execution of a computer program.

Objectives of a designed experiment include the following:

1. Isolate important factors from many other candidates
2. Identify better, more promising, operating conditions for each factor
3. Identify better combinations for test settings
4. Map the relation between input factors and output response
5. Identify significant interactions and curvature effects of the factors
6. Find factors which affect variability and settings which minimize variation.

The DEX approach is an empirical alternative to analytical approaches that are based on thorough understanding of the system. Many analytical approaches tend to lose effectiveness as the complexity of the system increases. This scaling property is important for parallel systems and their study. The statistical strategy can be applied to almost any system; it has been found to be effective even in the most complex multi-variable systems. Further, DEX is an inexpensive approach. Cost alone is enough reason to try DEX as an initial strategy.

The basic DEX approach is simple to describe, although in practice it can become complicated. Imagine a section of code, E, in a program. This factor E has two levels of treatment. The first is the original E, and the second a modified version of E (slightly different code). A given setting of E will occur in several trial runs in which other factors change. For analysis, all the runs with E in the second setting are averaged in response, and all runs with E in its original setting also are averaged. The difference between these two average responses is called a contrast for main factor E. (Note that in the main text, "effect" denotes a half-contrast or slope, i.e., the response difference divided by the input change of two.) A standard error (estimate of standard deviation) is used to keep or discard factors. Thus, if E's contrast is less than the standard error (noise, so to speak), there is a good chance that E is not very important; E is removed from further (refining) experiments. But note that this does not preclude E having an influence in conjunction with another code segment, say F. However, DEX analysis can also account for interactions of EF and a contrast can be obtained. Certain send-receive or lock-unlock code pairs in parallel programs may have such interactions. Software scaling discussed in the main exposition is a two-way interaction between (i) a piece of code and (ii) the size of the host system. These two-way scaling interactions may determine whether a parallel software package succeeds or fails.

7. Taylor's Expansion and DEX (Appendix B)

A Taylor series expansion of a function $f\,()$ is a power series [B1]. If $f^{i}\,()$ is the i^{th} derivative of $f\,() \equiv f^{0}()$, then $f(x)$ can be expressed in terms of f and its derivatives evaluated at some other point, say a. Given $(x-a)$ as the displacement from point a,

$$f(x) = \sum_{i=0}^{\infty} (x-a)^{i} \, \frac{f^{i}(a)}{i!}.$$

A more complex but essentially identical expansion holds for cases when f has multiple arguments. Thus for the multivariate response functions in DEX, Box, Hunter and Hunter remark that "the main effects and interactions can be associated with the terms of a Taylor series expansion of a response function. Ignoring, say, three-factor interactions corresponds to ignoring terms of third-order in the Taylor expansion" ([2], p.374). It is revealing to examine this remark in closer detail.

Let $R(x_1, x_2, x_3, \cdots, x_k)$ be a multivariate response function with k input variables corresponding to k factors of interest. The Taylor expansion of R is chosen about $a=0$, so that $(x-a)=x$. Such a choice centers the expansion amid settings of $\{x_j = \pm 1\}$, in correspondence to DEX convention:

$$R(\hat{x}) = R(\hat{0}) + \sum_{j=1}^{j=k} x_j \left[\frac{\partial R}{\partial x_j}\right]_0 + \tfrac{1}{2}\sum_{j=1}^{j=k} x_j^2 \left[\frac{\partial^2 R}{\partial x_j^2}\right]_0 + \sum_{i<j}^{k-1} \sum_{j=2}^{j=k} x_i\, x_j \left[\frac{\partial^2 R}{\partial x_i \partial x_j}\right]_0 + \cdots \qquad (i)$$

For screening experiments (including the scaling test), a two-level setting of factors is usually adequate; this implies that any second or higher order derivatives in the same variable are insignificant relative to the first-order derivatives. Here, they are thus assumed to be identically zero, *e.g.*, $\left[\partial^2 R / \partial x_j^{\,2}\right]_0 \equiv 0$ in equation (i) above. The term $R(\hat{0})$ in (i) corresponds in the DEX formulation to μ, the average response over all settings. Also note that the average DEX setting of the inputs X_i is zero; this stems from a deliberate design in DEX treatment patterns. The change of variable $X_i \leftarrow x_i$ is consequently straightforward. Derivatives that are not identically zero correspond to DEX β coefficients. Thus $\partial R / \partial x_j = \beta_j$ and similarly $\partial R^2 / \partial x_i \partial x_j = \beta_{i,j}$. The product $x_i\, x_j$ in equation (i) should look familiar, for it is the second-order interaction $I_{i,j} \equiv X_i\, X_j$ as defined in the DEX formulation. Applying all these substitutions in equation (i) produces a result that matches a standard DEX formulation:

$$R(\cdots) = \mu + \sum_{j=1}^{j=k} \beta_j X_j + \sum_{i<j}^{k-1} \sum_{j=2}^{j=k} \beta_{i,j} I_{i,j} + \cdots \qquad (ii)$$

Equation (ii) demonstrates that DEX analysis does capture expansion terms of a system's response function, as asserted by Box, Hunter and Hunter.

[B1] G. James and R.C. James. *Mathematics Dictionary,* (Van Nostrand Reinhold Company, 1976), 383-384.