John von Neumann Institute for Computing

Computational Force: A Unifying Concept for Scalability Analysis

Robert W. Numrich

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 107-112, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Computational Force: A Unifying Concept for Scalability Analysis

Robert W. Numrich

Minnesota Supercomputing Institute
University of Minnesota
Minneapolis, MN 55455 USA
E-mail: rwn@msi.umn.edu

Computational force, also called computational intensity, is a unifying concept for understanding the performance of parallel numerical algorithms. Dimensional analysis reduces a formula for execution time, from a paper by Stewart, to an exercise in differential geometry for a single efficiency surface. Different machines move on the surface along different paths defined by curvilinear coordinates that depend on ratios of hardware forces to software forces.

1 Introduction

In an early paper on the scalability of parallel machines[1], Stewart reported formulas for execution time for eight different algorithms. The one algorithm he described in detail yielded the formula,

$$t = \alpha n^3/p + 2n\sqrt{p}(\sigma + \tau) + (\sqrt{p} + mn)(\sigma + n\tau/(m\sqrt{p})) \; . \tag{1.1}$$

Table 3 of Stewart's paper displayed the quantities involved in his formula and the units he used to measure them. The quantity t was the execution time; α was the reciprocal of the computational power of a single processor; σ was a startup time; τ was the reciprocal of bandwidth; n was the matrix size; m was a block size; and p was the number of processors. He measured length in units of eight-byte words ($l_0 = 1$ word), work in the units of eight-byte floating-point operations ($e_0 = 1$ flop), and time in units of seconds ($t_0 = 1$ s). The details of the algorithm are not important for the purpose of this paper. The interested reader should consult Stewart's original paper.

2 Dimensional Analysis

Dimensional analysis is based on the algebraic properties of the matrix of dimensions[2–7]. In the system of units used by Stewart, the quantities involved in formula (1.1) have the matrix of dimensions,

	l_0	e_0	t_0	t	α	σ	τ	n	m	p
L(word)	1	0	0	0	0	0	−1	0	0	0
E(flop)	0	1	0	0	−1	0	0	0	0	0
T(s)	0	0	1	1	1	1	1	0	0	0

(2.1)

The entries for each quantity are the powers for each unit that correspond to its dimension. For example, reciprocal bandwidth τ is measured in units (s/word) with entries (-1,0,1), and reciprocal power α is measured in units (s/flop) with entries (0,-1,1).

From matrix (2.1), it is clear that formula (1.1) is not dimensionally consistent. Stewart's analysis, along with many others like it, lacks a clear distinction between the unit of length l_0 and the unit of work e_0. The problem size, which is dimensionless, enters into the calculation of both the number of operations performed and the number of words moved. Because the work grows like the cube of the problem size, the first term in formula (1.1) should be written as $(\alpha n^3/p)e_0$ to give it the correct dimension of time. Furthermore, adding together the startup time σ and the reciprocal bandwidth τ makes no sense unless we multiply τ by the length of the transfer. In the first appearance of the quantity τ in formula (1.1), the length is presumably a single word of length l_0, and in the second appearance, it is nl_0.

To make the formula dimensionally consistent and to simplify it for the next step in the analysis, we define the square root of the number of processors,

$$q = \sqrt{p} \, , \tag{2.2}$$

and the work done per processor,

$$w = n(n/q)^2 e_0 \, . \tag{2.3}$$

We also define the dimensionless quantity,

$$s = n + 2nq + q \, , \tag{2.4}$$

and the length,

$$l = (2nq + (n/q)(n+q))l_0 \, , \tag{2.5}$$

noting that Stewart sets the parameter $m = 1$. Execution time then obeys the formula,

$$t = \alpha w + s\sigma + l\tau \, , \tag{2.6}$$

Rewriting the formula in correct dimensional form is not a pedantic quibble. It is an essential step in dimensional analysis[2–7].

We need, therefore, to consider execution time as a function of six dependent variables,

$$t = t(\alpha, w, s, \sigma, l, \tau) \, . \tag{2.7}$$

For the next step in the analysis, we switch to a system of measurement based on length, force, and time as primary units. Computational force, measured in the unit,

$$f_0 = e_0/l_0 = 1 \text{ flop/word} \, , \tag{2.8}$$

is also called computational intensity[8–12]. It is recognized as an important quantity in performance analysis, and we show, in what follows, that it provides a unifying concept for scalability analysis. In this new system of units, the quantities involved have the matrix of dimensions,

$$\begin{array}{c|ccc|cccccccc} & l_0 & f_0 & t_0 & t & \alpha & w & \sigma & \tau & l & s \\ \hline L(\text{word}) & 1 & 0 & 0 & 0 & -1 & 1 & 0 & -1 & 1 & 0 \\ F(\text{flop/word}) & 0 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ T(\text{s}) & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{array} \tag{2.9}$$

The fundamental assumption of dimensional analysis is that relationship (2.7) is independent of the system of units we use[2–7]. Under this assumption, we may pick three

parameters, $\alpha_L, \alpha_F, \alpha_T$, one for each primary unit, and scale each quantity in the relationship, using the entries in the matrix of dimensions (2.9), such that

$$\alpha_T t = t(\alpha_L^{-1}\alpha_F^{-1}\alpha_T\alpha, \alpha_L\alpha_F w, s, \alpha_T\sigma, \alpha_L l, \alpha_L^{-1}\alpha_T\tau) \; . \tag{2.10}$$

We can pick these scale factors in such a way to establish a new system of units where the number of parameters reduces from six to three and all quantities are scaled to dimensionless ratios. Indeed, if we pick them such that

$$\alpha_L^{-1}\alpha_F^{-1}\alpha_T\alpha = 1 \; , \qquad \alpha_L\alpha_F w = 1 \; , \qquad \alpha_L l = 1 \; , \tag{2.11}$$

we can solve for the three parameters to find

$$\alpha_L = 1/l \; , \qquad \alpha_F = l/w \; , \qquad \alpha_T = 1/(\alpha w) \; . \tag{2.12}$$

In this new system of units, formula (2.10) becomes

$$t/(\alpha w) = t(1, 1, s, \sigma/(\alpha w), 1, l\tau/(\alpha w)) \; . \tag{2.13}$$

We recognize the time,

$$t_* = \alpha w \; , \tag{2.14}$$

as the minimum time to complete the computation. The left side of formula (2.13) is, therefore, the reciprocal of the efficiency,

$$t/t_* = 1/e \; . \tag{2.15}$$

Scaling (2.6) with the parameters from (2.12), we find the efficiency function,

$$e = [1 + s\sigma/(\alpha w) + l\tau/(\alpha w)]^{-1} \; . \tag{2.16}$$

3 Computational Forces and the Efficiency Surface

The efficiency formula (2.16) depends on ratios of computational forces. To recognize this fact, we first observe that the efficiency formula assumes the simple form,

$$e = \frac{1}{1 + u + v} \; , \tag{3.1}$$

in terms of the two variables,

$$u = s\sigma/(\alpha w) \qquad v = l\tau/(\alpha w) \; . \tag{3.2}$$

Each of these variables is the ratio of opposing computational forces. Using the quantities defined by equations (2.3)-(2.5), we find the first variable,

$$u(n, q) = \frac{\phi_1^H}{\phi_1^S(n, q)} \; , \tag{3.3}$$

is the ratio of a hardware force,

$$\phi_1^H = (\sigma/l_0)/\alpha \; , \tag{3.4}$$

to a software force,

$$\phi_1^S = \frac{n(n/q)^2}{n + 2nq + q} f_0 \; . \tag{3.5}$$

In the same way, the second variable is the ratio of two other opposing forces,

$$v(n,q) = \frac{\phi_2^H}{\phi_2^S(n,q)}, \tag{3.6}$$

with a second hardware force,

$$\phi_2^H = \tau/\alpha \ , \tag{3.7}$$

and a second software force,

$$\phi_2^S(n,q) = \frac{n(n/q)^2}{2nq + (n/q)(n+q)} f_0 \ . \tag{3.8}$$

The two hardware forces are independent of the problem size and the number of processors. The first one is determined by the ratio of the startup time to the reciprocal of the computational power, and the second one is determined by the ratio of the reciprocal of the bandwidth to the reciprocal of the computational power. The two software forces, on the other hand, are functions of problem size and the number of processors, independent of the hardware. They are determined by this particular algorithm.

For fixed problem size, $n = n^*$, the curvilinear coordinates,

$$u^*(q) = u(n^*, q) \qquad v^*(q) = v(n^*, q) \ , \tag{3.9}$$

define paths along the surface,

$$e^*(q) = \frac{1}{1 + u^*(q) + v^*(q)} \ , \tag{3.10}$$

as the number of processors changes. Similarly, for fixed number of processors, $q = q_*$, the curvilinear coordinates,

$$u_*(n) = u(n, q_*) \qquad v_*(n) = v(n, q_*) \ , \tag{3.11}$$

define paths along the surface,

$$e_*(n) = \frac{1}{1 + u_*(n) + v_*(n)} \ , \tag{3.12}$$

as the problem size changes. Different machines follow different paths determined by their particular values of the hardware forces ϕ_1^H and ϕ_2^H.

4 Discussion

If we use the same numerical algorithm, the software forces have not changed in the decade and a half since the publication of Stewart's paper. But the hardware forces have changed as shown in the following table,

	Machine 1 ca. 1990	Machine 2 ca. 2007
$1/\alpha$	10^6 flop/s	10^9 flop/s
σ	10^{-4} s	10^{-6} s
$1/\tau$	10^6 word/s	10^9 word/s
ϕ_1^H	10^2 flop/word	10^3 flop/word
ϕ_2^H	1 flop/word	1 flop/word

(4.1)

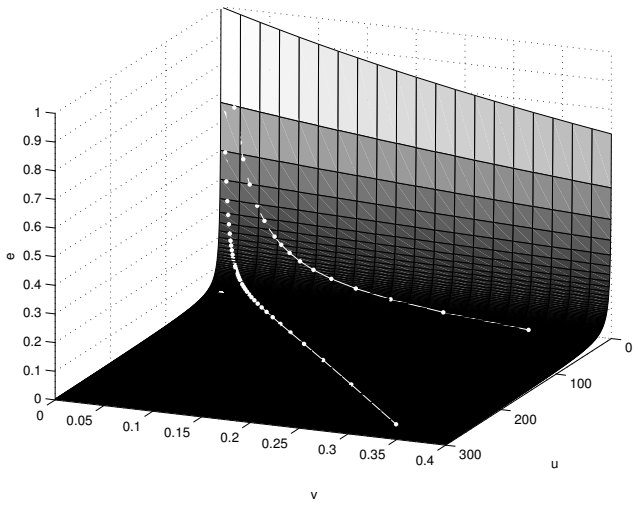

Figure 1. Two paths along the efficiency surface for fixed number of processors, $p = q^2 = 32^2 = 1024$. The higher path corresponds to the older Machine 1; the lower path to the modern Machine 2. The problem size increases from $n = 500$ at low efficiency in increments of 100. In the limit of very large problem size, both machines approach perfect efficiency. They approach the limit at different rates along different paths on the surface.

Because of these changes, modern machines follow paths lower on the efficiency surface than paths followed by older machines.

Fig. 1 shows two paths along the efficiency surface, one for each machine in Table 4.1 for a fixed number of processors, $p = q^2 = 32^2 = 1024$. Each point on a path $e_*(n)$, calculated from (3.12) using the curvilinear coordinates $u_*(n)$ and $v_*(n)$ from (3.11), corresponds to a different problem size. For large enough problem, both machines approach perfect efficiency. But they follow different paths at different rates along the surface to reach the summit. The modern machine follows a path lower on the efficiency surface at all values of the problem size and is less efficient than the older machine.

The reason for this loss of efficiency is clear. The modern machine has higher values for the coordinate $u_*(n)$ for each problem size. It travels a long way across the surface at low efficiency before mounting the ascent to the summit. The coordinate $u_*(n)$ depends on the hardware force ϕ_1^H, which depends on the startup time σ. As Table 4.1 shows, this force has increased in the last two decades. It is harder now to overcome long startup times than it was before. This difficulty translates directly into a harder job for the programmer who must provide a larger software force ϕ_1^S to overcome the larger hardware force. To increase the software force, the programmer must design a new algorithm for the problem.

5 Summary

Our approach to performance analysis is quite different from the more familiar approaches that have appeared in the literature. Our first impulse is to plot the efficiency surface, or

contours of the surface, as a function of the problem size and the number of processors. With that approach, each machine has its own efficiency surface depending on its particular values of the hardware forces ϕ_1^H and ϕ_2^H. It is difficult to compare machines because they lie on different surfaces.

Our approach produces one efficiency surface for all machines. Different machines follow different paths along the same surface depending on their hardware forces ϕ_1^H and ϕ_2^H. The distance between paths, along the geodesic on the surface, measures the distance between machines. Two machines with the same hardware forces are self-similar and scale the same way even though the particular values for their startup time σ, their bandwidth τ^{-1}, and their computational power α^{-1} may be quite different. Only the forces determined by the ratios of these quantities matter not their absolute values. These forces provide a unifying concept for scalability analysis.

Acknowledgement

The United States Department of Energy supported this research by Grant No. DE-FG02-04ER25629 through the Office of Science.

References

1. G. W. Stewart, *Communication and matrix computations on large message passing systems*, Parallel Computing, **16**, 27–40, (1990).
2. G. I. Barenblatt, *Scaling*, Cambridge University Press, (2003).
3. G. Birkhoff, *Hydrodynamics: A Study in Logic, Fact and Similitude*, Princeton University Press, 2nd edition, (1960).
4. L. Brand, *The Pi Theorem of Dimensional Analysis*, Arch. Rat. Mech. Anal., **1**, 35–45, (1957).
5. P. W. Bridgman, *Dimensional Analysis*, 2nd ed., (Yale Univ. Press, 1931).
6. R. W. Numrich, *A note on scaling the Linpack benchmark*, J. Parallel and Distributed Computing, **67**, 491–498, (2007).
7. R. W. Numrich, *Computer Performance Analysis and the Pi Theorem*, (submitted 2007).
8. D. Callahan, J. Cocke and K. Kennedy, *Estimating Interlock and Improving Balance for Pipelined Architectures*, J. Parallel Dist. Comp., **5** 334–358, (1988).
9. R. W. Hockney, *Performance parameters and benchmarking of Supercomputers*, Parallel Computing, **17**, 1111–1130, (1991).
10. R. W. Hockney, *The Science of Computer Benchmarking*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, (1996).
11. J. D. McCalpin, *Memory Bandwidth and Machine Balance in Current High Performance Computers*, IEEE Computer Society; Technical committee on Computer Architecture Newsletter, (December 1995).
12. D. Miles, *Compute Intensity and the FFT*, in: Proc. Supercomputing 1993, pp. 676–684, (1993).