

Performance Metrics Based on Computational Action ^{*†}

Robert W. Numrich ^{‡§}

Abstract

We propose a new performance metric based on computational action. We examine work as it evolves in time and compute computational action as the integral of the work function over time. We compare the action generated at less than full power with the action that could have been generated at full power. We claim that the goal of performance optimization is to minimize lost, or wasted, action. We calculate our metric for some computers in the Top500 list and propose a new ranking based on least action wasted. When work is a function of the resources applied, we use the classical techniques of the calculus of variations to minimize wasted action. From the result of this exercise, we calculate productivity as the ratio of work produced to resources used.

1 Introduction

We consider the problem of measuring performance and productivity for projects that include both a software development phase and a code execution phase. Can we define new ways to look at resource management during such a project? What can we measure and what metric should we use to report the result of our measurements? We define a new metric based on the product of work and time, a quantity in physics, like Planck's constant, which has properties of action. We propose a principle of computational least action.

To measure anything we need a system of measurement and a system of units. We define such a system in Section 2.

In Section 3 we consider the problem of optimizing time. We discuss a simple hypothetical project in terms of time charged to the project subject to constraints imposed on the project.

In Section 4 we change focus from optimizing time to optimizing work. A project must do some work, must deliver some product, and must do so by using the resources available. So we look at the constraints imposed on the work to meet the project's goals.

^{*}To appear: International Journal of High Performance Computing Applications, Special Issue Vol. 18, No. 3, Fall 2004

[†]This research was supported in part by grant DE-FC02-01ER25505 from the U.S. Department of Energy as part of the Center for Programming Models for Scalable Parallel Computing sponsored by the Office of Science. It was also supported in part by a consulting contract from SGI through the Defence Advanced Research Projects Agency (DARPA) under Contract No. NBCHC-02-0054. And it was supported in part by the NASA Goddard Earth Sciences and Technology Center where the author holds an appointment as a Goddard Visiting Fellow for the academic year 2003-2004.

[‡]Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN 55455, USA

[§]rwn@msi.umn.edu

In Section 5 we change focus again to look at the combination of work and time together. We define a metric that measures lost or wasted action and propose a principle of least computational action. We compute the metric for some computers on the Top500 list.

In Section 6 we apply the classical methods of the calculus of variations to a simple problem to show how to minimize wasted action.

In Section 7 we apply the results of our exercise in the calculus of variations to define and calculate a productivity metric. This metric is the ratio of the work produced to the resources used.

In Section 8 we summarize and recapitulate our results.

2 Units and Systems of Measurement

To define metrics, we need a system of measurement; to define a system of measurement, we need to know what we are trying to measure; to measure, we need a system of units. The international physics community, after centuries of confusion, has agreed on the System International system of units [1]. The computational science community, on the other hand, has not defined a system of units.

Table 1: Fundamental Computational Quantities

Name	Unit	Value
binary operation	op	1 op
bit	b	1 b
machine frequency	Hz	1 s^{-1}
floating-point operation	flop	64 op
byte	byte	8 b
kilobyte	kbyte	10^3 byte
megabyte	Mbyte	10^6 byte
megaflop	Mflop	10^6 flop
megahertz	MHz	10^6 Hz

An important unit is the unit of time. We suggest that the unit of time be defined as one period $\tau = \nu^{-1}$ where ν is a fundamental frequency defining the system. For example, if a computer's operating frequency is 1 MHz, the unit of time is $1 \mu\text{s}$. Time measured in seconds becomes the dimensionless clock-tick, $k = \nu t$, when multiplied by the frequency. If different parts of the same machine run at different frequencies or if we need to compare different machines running at different frequencies, then we need to pick one frequency to define our unit of time and convert all others to the same unit. For example, if $k_1 = \nu_1 t$, we must convert it to $k_1 = (\nu_1/\nu)\nu t = (\nu_1/\nu)k$.

Another important unit is the unit of length. An obvious choice for the unit of length is the binary bit. This choice is implied by words we use to describe sizes like bits, bytes, words, double words, and so forth. We suggest that the fundamental unit of length be a single bit with symbol b. Other units of length are derived from this unit by multiplying by constant factors, for example, 1 byte = 8 b, 1 word = 64 b.

To define a consistent set of units, we need a third fundamental quantity. Since we have no intuitive idea

for something like mass, which is used to define the system of measurement in mechanics, we pick our third fundamental quantity to be work. We suggest that the fundamental unit of work be the binary operation, in other words, the production of one bit of information either a one or a zero. The fundamental unit is represented by the symbol *op*. Different kinds of work may have different names, for example, floating-point work, integer work or logical work. The kind of work that is important is different for different applications. If the same number of bits are produced the amount of work is the same independent of the kind of work. It may take more or less time to produce the same number of bits, so the power, the rate of doing work, is different, but the amount of work is the same.

Table 1 summarizes our fundamental system of units for computational performance measurement [8]. It also lists some convenient multiples of the fundamental units, which we commonly use. We use the SI definitions [1] for prefixes such as $M=10^6$ for the prefix mega. In Section 5, we use $Z=10^{21}$ for the prefix zetta. Using these prefixes to mean powers-of-two [9] creates unnecessary conflict with the rest of the scientific community. The symbol $k=10^3$ is the correct prefix for kilo; the symbol *K* is reserved for the unit of temperature to honor Lord Kelvin. Symbols for units are capitalized only when they are based on a person’s name. The unit for one byte, therefore, is not capitalized.

Table 2: Fundamental and Derived Computational Quantities

Name	Symbol	Value	Unit
work	ω	ω	op
length	β	β	b
frequency	ν	ν	Hz
clock period	τ	ν^{-1}	s
clock tick	k	νt	-
time	t	k/ν	s
power	r	$\omega\nu$	op/s
action	S	ω/ν	op·s

With this system of fundamental units, what derived quantities can we measure in that system [2, 3]. Table 2 lists some possibilities. In this paper we pay particular attention to the derived quantity called action. It has units of work multiplied by time, for example, *flop*·s. This quantity is different from power, which has units of work divided by time, for example, *flop*/s. The two quantities are quite different and should not be confused. Although power is a commonly reported performance metric, action is a new metric.

3 Time

Increasing productivity by minimizing the “total-time-to solution” is a somewhat ill-defined statement of the problem. We adopt an alternative statement: At each moment in time, use the resources available in an optimal way to accomplish a mission within imposed constraints. Such a statement suggests application of the methods of the calculus of variations, which we develop in Section 6.

Before we can develop a mathematical formulation of the somewhat fuzzy idea of optimizing the total-

time-to solution, we need a precise definition of “time.” For example, we might mean real time, sometimes referred to as wall-clock time, for which we use the symbol t . Or we might mean CPU time, the accumulated processing time used on a computing system, which is charged to a project according to some accounting algorithm. Or we might mean employment time for programmers writing code, which is charged to the project as wages. To make a clear distinction between kinds of time, we let the function $T = T(t)$ be the accumulated time charged to a project at real time t . The function $T(t)$ includes all aspects of the project, not just computer time.

There are at least two constraints on any project. It must deliver some product so it must accumulate some minimum amount of time, say $T_{\min}(t)$. On the other hand, it cannot accumulate more time than its budget can afford, say $T_{\max}(t)$. So an obvious constraint is

$$T_{\min}(t) \leq T(t) \leq T_{\max}(t) . \quad (3.1)$$

For simplicity, suppose there are two kinds of resources used in the project, for example, software development resources and computer system resources. Each piece accumulates time according to its own function of time, $t_1 = t_1(t)$ and $t_2 = t_2(t)$. The total accumulated time charged to the project is the sum

$$T(t) = t_1(t) + t_2(t) . \quad (3.2)$$

At first glance, the function $T(t)$ appears to be a linear function of its two pieces, but it is, in fact, highly nonlinear. A program runs faster if we put more time into software development. If a program does not perform well, we go back and do more coding. In other words, we have a nonlinear feedback loop in the software development process. The accumulated time charged to the project should rather be written something like

$$T(t) = t_1(t_2(t), t) + t_2(t_1(t), t) \quad (3.3)$$

to reflect the fact that development time and execution time depend on each other in a very complicated way.

To illustrate the behavior we might expect, consider piece-wise linear functions for the two pieces of the function $T(t)$. Figure 1 shows how time might accumulate over the lifetime of a hypothetical project. A typical project alternates between code development time and code execution time. At the beginning of a project, all the effort may go into code development with no code execution until the first software release. The function $t_1(t)$, therefore, increases at a rate determined by how quickly developers can produce code while the function $t_2(t)$ remains flat.

At the first software release, code development stops, the function $t_1(t)$ remains flat, and the execution time $t_2(t)$ starts to increase as the project accumulates computer charges. At some point, feedback about how well the code runs causes the development team to write new code, causing the function $t_1(t)$ to rise again. The function $t_2(t)$ continues to rise at the same rate determined by the first version of the code. At the time of the second software release, the function $t_1(t)$ goes flat again while the function $t_2(t)$ accumulates time at a different rate because the code, presumably, gets the same amount of work done in less time.

This cycle between code development and code execution continues, with feedback between the two pieces, until it no longer makes sense, for one reason or another, to spend more development effort. At that time, the function $t_1(t)$ goes flat while the function $t_2(t)$ continues to rise at a rate determined by the final software release. The project continues in this mode until it completes all the work it was assigned. The two constraints on time from (3.1) are represented by the upper dotted line in Figure 1 for $T_{\max}(t)$ and by the lower dotted line for $T_{\min}(t)$.

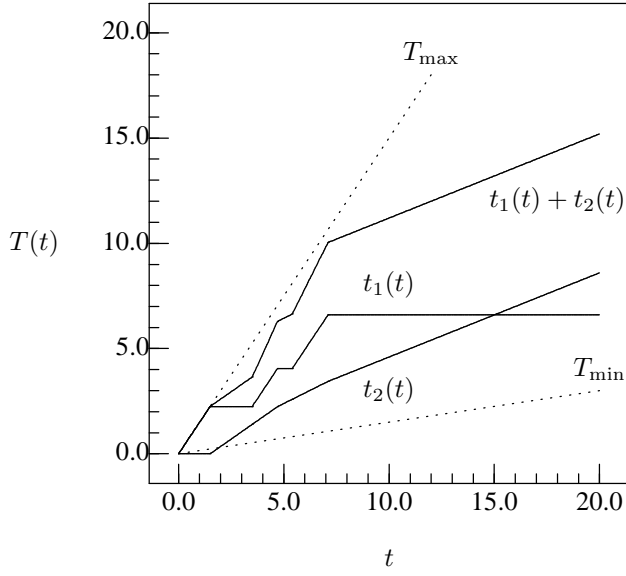


Figure 1: Accumulated time charged to the project as a function of time.

Eventually time is converted to money. We convert development time into money using the hourly wage of the development staff, say b_{dev} , so that the cost of development increases in time by the function

$$c_1(t) = b_{\text{dev}}(t) \times t_1(t) . \quad (3.4)$$

In the same way, we can convert execution time into money using the charge for computer time, say b_{comp} , so that the operational cost increases in time by the function

$$c_2(t) = b_{\text{comp}}(t) \times t_2(t) . \quad (3.5)$$

The expenditure function for the project is the sum of these two functions,

$$E(t) = c_1(t) + c_2(t) . \quad (3.6)$$

We want to spend money in an optimal way using the resources at our disposal while getting the work done within the project deadline. What is considered optimal may depend on external criteria that may have little to do with the actual deliverables of the project. In good times, we may want to spend all the money allotted to the project, which we represent by the function $B_{\text{max}}(t)$. In that case, we might want to minimize the integral,

$$J = \int_0^{t^*} |B_{\text{max}}(t) - E(t)| dt , \quad (3.7)$$

subject, of course, to the constraint that the work is completed within a specified deadline represented by the time t^* . In bad times, on the other hand, budgets may be low and we may need to cut costs. In that case, we might want to minimize the integral,

$$J = \int_0^{t^*} |B_{\text{min}}(t) - E(t)| dt , \quad (3.8)$$

where $B_{\min}(t)$ is the smallest budget required to finish the project.

The key to understanding what it means to optimize the total-time-to-solution is to understand the work that must be done to finish the project, what resources are needed to accomplish the work, and the constraints that must be satisfied. In the next section, we turn our attention away from looking at time itself and toward the work that accumulates as the project advances. At the end of the project, the work should be complete and the constraints satisfied.

4 Computational Work

In this section, we shift our attention away from time itself to work as it evolves in time. A project must deliver a product. To deliver that product, the project must do some work. We can name our unit of work anything we want, but it must be something we can measure. For example, a floating-point operation, as defined in Table 1, might be appropriate for measuring computer work. Writing a line of code might be appropriate for measuring programmer work but there is little agreement on how to define this kind of work. Different kinds of work may be appropriate for different parts of a project. We define the unit of work to be ω and there must be a way to convert different kinds of work to this unit. How to make these conversions is the topic of current research.

As time increases, work accumulates. If the time scale is determined by the reciprocal of a frequency ν , and if the project is able to produce one unit of work each time period, then the project can deliver the maximum power,

$$r_0 = \omega\nu . \quad (4.1)$$

Work as a function of time t accumulates, at most, linearly like the function

$$W_0(t) = r_0 t . \quad (4.2)$$

If the project must complete a fixed amount of work, W_* , at full power it will finish at time t_* determined by setting the left side of equation (4.2) equal to W_* ,

$$W_* = r_0 t_* , \quad (4.3)$$

and solving for the minimum time,

$$t_* = W_*/r_0 . \quad (4.4)$$

Higher power, resulting from either a higher frequency or a higher unit of work, yields a lower minimum time.

Real projects do not run at full power. Work accumulates at a slower rate according to some function $W(t)$ such that

$$W(t) \leq W_0(t) . \quad (4.5)$$

The project completes at time t^* when

$$W(t^*) = W_* . \quad (4.6)$$

The time t^* is greater than the minimum time t_* . In fact, from equations (4.2) and (4.5), we have

$$W(t^*) \leq r_0 t^* , \quad (4.7)$$

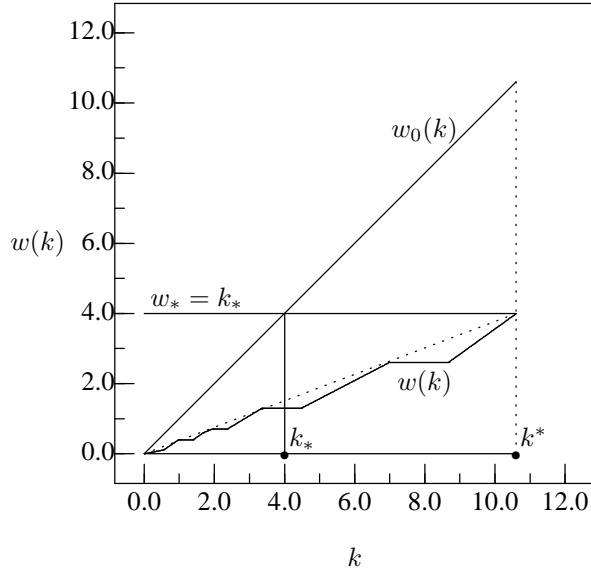


Figure 2: Accumulated work as a function of time.

and from equations (4.3) and (4.6), we have

$$r_0 t_* = W(t^*) . \quad (4.8)$$

Substituting the last equation into inequality (4.7), we obtain

$$r_0 t^* \geq r_0 t_* \quad (4.9)$$

so that

$$t^* \geq t_* , \quad (4.10)$$

as we claimed.

Figure 2 shows how work accumulates with time. We have plotted the dimensionless work function

$$w(k) = W(k/\nu)/\omega \quad (4.11)$$

as a function of the dimensionless clock-tick

$$k = \nu t . \quad (4.12)$$

At full power, work accumulates along the diagonal line since

$$w_0(k) = W_0(k/\nu)/\omega = \frac{\omega\nu(k/\nu)}{\omega} = k , \quad (4.13)$$

and the project stops when $k_* = w_*$ where $w_* = W_*/\omega$.

Work cannot accumulate faster than the diagonal line so the actual work function $w(k)$ looks something like the wiggly line below the diagonal. The project completes when

$$w(k^*) = w_* . \quad (4.14)$$

Work accumulates faster or slower at different times during the project. Flat spots in the curve, where no work accumulates, might correspond to a delay in a program waiting for data to arrive or to a delay in software development caused by an unexpected redesign of the code.

Without detailed knowledge of the work function, we often focus on a single point in the work-time plane and compute the average power,

$$r = \frac{\omega w_*}{k_*^*/\nu} = \omega \nu \frac{k_*}{k_*^*} = r_0 \frac{k_*}{k_*^*}. \quad (4.15)$$

The efficiency ϵ , therefore, is the slope of the dotted line in Figure 2,

$$\frac{r}{r_0} = \epsilon = \frac{k_*}{k_*^*}. \quad (4.16)$$

This metric is often quoted, for example, in the Linpack benchmark [10].

As is clear from the figure, however, when we compute average behavior, we lose all the specific detail of what happened between the start of the project and the end of the project. All the interesting detail is in the wiggles. Average behavior tells us nothing about why we are not performing at full power. Without knowing what is happening, we have no way to know how to fix the problem. For this reason, we suggest, in the next section, that we look at the information in a different way.

5 Computational Action

In this section, we shift focus once again, away from work alone or time alone, to consider action, the product of work and time. The area of any part of the plane in Figure 2 has units of work-by-time or units of action. In particular, the integral of the work function,

$$S(t) = \int_0^t W(\tau) d\tau, \quad (5.1)$$

measures the action generated at time t .

We define the dimensionless function,

$$s(k) = (\nu/\omega) S(k/\nu). \quad (5.2)$$

After the change of variable, $\kappa = \nu\tau$, we find

$$s(k) = \int_0^k w(\kappa) d\kappa, \quad (5.3)$$

the dimensionless action at clock-tick k .

Action is bounded above and below by

$$s_* \leq s(k^*) \leq s^* \quad (5.4)$$

where

$$s_* = \int_0^{k_*} w_0(\kappa) d\kappa = k_* k_*/2 \quad (5.5)$$

is the action generated at full power up to clock-tick k_* , and

$$s^* = \int_0^{k_*^*} w_0(\kappa) d\kappa = k_*^* k_*/2 \quad (5.6)$$

is the action generated at full power up to clock-tick k^* . The first inequality follows by definition since s_* is the minimum action that can be generated. The second inequality follows from inequality (4.5).

If we approximate the generated action by the area of the dotted triangle in Figure 2,

$$s(k^*) \approx k^* k_*/2, \quad (5.7)$$

then we have

$$k_* k_* \leq k^* k_* \leq k^* k^*. \quad (5.8)$$

From the first inequality, we find

$$\epsilon = \frac{k_*}{k^*} \leq 1 \quad (5.9)$$

while from the second inequality

$$\frac{1}{\epsilon} = \frac{k^*}{k_*} \geq 1. \quad (5.10)$$

This approach yields nothing more than we already knew from efficiency.

Another approach, which yields something new, is suggested by rewriting the action integral as the sum

$$s(k) = \int_0^k w_0(\kappa) d\kappa - \int_0^k (w_0(\kappa) - w(\kappa)) d\kappa. \quad (5.11)$$

The second integral represents lost, or wasted, action, We are encouraged, therefore, to minimize the integral,

$$I(k) = \int_0^k (w_0(\kappa) - w(\kappa)) d\kappa. \quad (5.12)$$

We can approximate this integral by using equations (5.6) and (5.7) such that

$$I(k^*) \approx \frac{k^* k^* - k^* k_*}{2}. \quad (5.13)$$

Multiplying this approximation by the unit of action, ω/ν , we suggest the following metric

$$\eta = \frac{k^* (k^* - k_*)}{2} \cdot (\omega/\nu), \quad (5.14)$$

as a measure of lost, or wasted, computational action. The lower the value of this metric, the more effectively the resources have been used.

It is possible to compute this new metric for some, but not all, of the computers in the Top500 list [10]. Some of the entries in the list do not contain enough information to make the calculation. Table 3 shows the result for selected entries from the list for June 2003. It shows the original ranking, based on highest power, and the new ranking, based on the lost action metric. Figure 3 displays the same data. The fact that the rankings are shuffled indicates that the new metric contains different information. It may be a better measure of a machine's productivity, but it is beyond the scope of this paper to suggest possible reasons for the new ranking compared to the old.

Calculation of the new metric requires all the data reported in the Top500 report, not just the maximum power. It also requires that the problem size for each entry be the same, which means the amount of work is the same. This requirement represents a major difference between the two metrics because each entry in the Top500 report does a different amount of work. We first had to put them on the same footing by estimating the time for each system to solve a fixed-size problem, which we arbitrarily picked as $n = 10^6$. We assumed the total work for this size problem to be $W_*(n) = 2n^3/3$ and that each machine requires time

$t = t_0 + W_*(n)/R_{\text{peak}}$ to complete the work. From the reported values of R_{max} and N_{max} , we computed the time $t = 2N_{\text{max}}^3/(3R_{\text{max}})$. Using this calculated time along with the reported value for R_{peak} and $W_*(N_{\text{max}}) = 2N_{\text{max}}^3/3$, we computed t_0 . We then had enough information to compute the metric η as reported in Table 3. Whether our assumptions on the model for computation time for fixed work is correct or not needs to be investigated further but is beyond the scope of this paper.

Another approach to the problem of optimal resource allocation is represented by the ESP benchmark [12]. That benchmark considers the derivative of the accumulated work function,

$$r(t) = dW/dt , \quad (5.15)$$

which is power at time t . They compute the integral

$$ESP(t) = \int_0^t r(\tau)d\tau . \quad (5.16)$$

For a piece-wise linear problem, the power curve is a step function equal to zero during periods when no work is accumulating and equal to a constant during periods when work is accumulating at a fixed rate. The area under such a curve equals the total work done as a function of time. The relationship with our action metric is the double integral

$$S(t) = \int_0^t \int_0^\sigma r(\tau)d\tau d\sigma . \quad (5.17)$$

In an earlier approach to performance analysis, Hack [5] considered the integral

$$H = \int_0^1 W(\sigma)d\sigma \quad (5.18)$$

where σ was an abstract dimensionless variable of integration so that his integral, like the ESP integral, has units of work.

6 Computational Least Action

In this section, we apply the methods of the calculus of variations [4] to find an optimal work function that minimizes lost action. This is a new application of what we have called the *Principle of Computational Least Action* [7]. Tyagi [11] states a different principle of least computational action as applied to VLSI circuit design.

We minimize the functional

$$J = \int_0^{t^*} (W(p, \dot{p}) - W_0(t))^2 dt \quad (6.1)$$

where work $W(p, \dot{p})$ is a function of the unknown function $p(t)$ and its derivative with respect to time \dot{p} . The function $p(t)$ determines how resources are applied to perform the required work. We want to find the function $p(t)$ that minimizes the functional J subject to appropriate boundary conditions and appropriate constraints. An obvious constraint is that we must get the work done before we stop. Hence we must satisfy the constraint

$$W(p(t^*), \dot{p}(t^*)) = W_* , \quad (6.2)$$

which determines the time t^* when the project is finished.

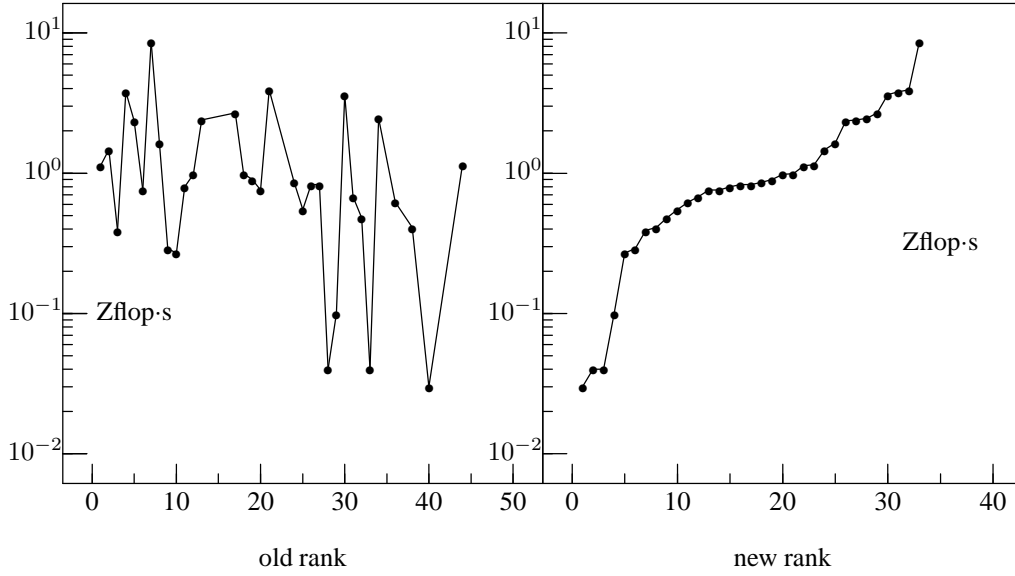


Figure 3: Lost action for selected computers from the Top500 list for June 2003. The value of the new metric is in units of Zflop·s, 10^{21} floating-point operations multiplied by the time in seconds. The value is large because the problem size was picked arbitrarily as 10^6 .

6.1 A Linear Work Model

For a linear work model,

$$W(p, \dot{p}) = ap + b\dot{p}, \quad (6.3)$$

it is possible to allocate resources as a function of time such that work accumulates at the maximum rate. The parameter $a > 0$ has dimension of work and the parameter $b > 0$ has dimension of work-by-time. Their ratio defines a natural frequency for the problem

$$\nu = a/b. \quad (6.4)$$

The reciprocal of the frequency sets a time scale for the system, and the parameter a is a natural unit of work.

If the maximum power of the system is $r_0 = a\nu$, then work accumulates no faster than the function

$$W_0(t) = a\nu t. \quad (6.5)$$

The objective function (6.1) for this problem is the integral

$$J = a^2 \int_0^{t^*} [p + \nu^{-1}\dot{p} - \nu t]^2 dt. \quad (6.6)$$

To find the optimal function $p(t)$ we need to solve the second order Euler differential equation

$$\nu^{-2} \cdot \ddot{p} = p + (1 - \nu t). \quad (6.7)$$

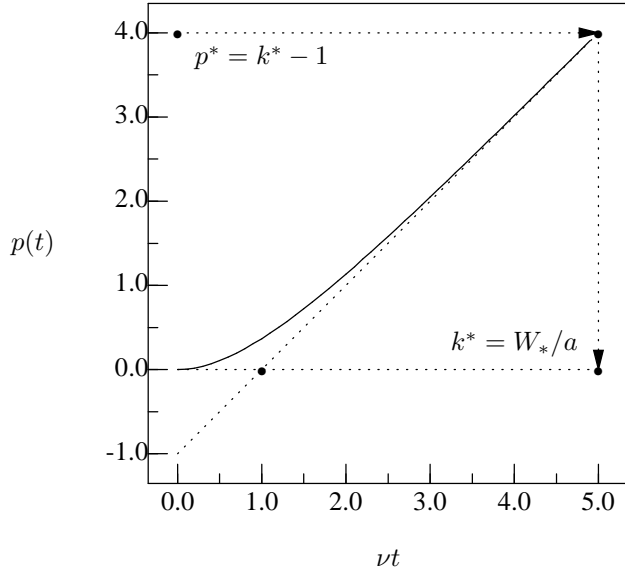


Figure 4: Accumulated resources $p(t)$ as a function of time.

If we assume no resources applied and no work done at time zero, the appropriate initial conditions are

$$p(0) = 0 \quad \dot{p}(0) = 0 . \quad (6.8)$$

The solution to this differential equation is the function

$$p(t) = \exp(-\nu t) + \nu t - 1 . \quad (6.9)$$

Its first derivative is the function

$$\dot{p}(t) = -\nu \exp(-\nu t) + \nu \quad (6.10)$$

and the work function becomes

$$W(t) = a (\exp(-\nu t) + \nu t - 1) + b\nu (1 - \exp(-\nu t)) . \quad (6.11)$$

After observing from definition (6.4) that $b\nu = a$, we find the work function

$$W(t) = a\nu t , \quad (6.12)$$

which equals the maximum work function (6.5). We confirm, therefore, that resource allocation according to the function $p(t)$ of equation (6.9) yields maximum power.

Figure 4 shows the function $p(t)$. It applies to all systems with the same work function given by definition (6.9) with the unit of work for each particular system fixed by the parameter a and the time scale fixed by the reciprocal of the frequency ν . For $t > \nu^{-1}$, the function $p(t)$ asymptotically approaches the straight line,

$$p_a(t) = \nu t - 1 . \quad (6.13)$$

The total time t^* to complete the project is determined by equation (6.12),

$$W_* = avt^* . \quad (6.14)$$

In clock-ticks, the project is complete at

$$k^* = W_*/a \quad (6.15)$$

and the total resources expended to perform the work is

$$p^* = k^* - 1 . \quad (6.16)$$

In actual time, the project completes at

$$t^* = \frac{bW_*}{a^2} , \quad (6.17)$$

longer or shorter depending on the parameters defining the system.

6.2 A Nonlinear Work Model

A more interesting, but more difficult, example is generated by nonlinear work models. For example, suppose the work function has the form

$$W(p, \dot{p}) = ap^2 + b\dot{p}^2 . \quad (6.18)$$

The parameter $a > 0$ has dimensions of work and is again the natural unit of work. The parameter $b > 0$ has dimensions of work by time-squared so the characteristic frequency for this system is the frequency

$$\nu = \sqrt{a/b} . \quad (6.19)$$

The objective function (6.1) for this problem is the integral

$$J = a^2 \int_0^{t^*} [p^2 + \nu^{-2}\dot{p}^2 - \nu t]^2 dt \quad (6.20)$$

with the boundary conditions

$$p(0) = 0 \quad \dot{p}(0) = 0 . \quad (6.21)$$

The second order Euler differential equation is the equation

$$2\nu^{-4}\dot{p}^2\ddot{p} = p^3 - \nu tp - \nu^{-2}p\dot{p}^2 + \nu^{-1}\dot{p} . \quad (6.22)$$

This highly nonlinear equation must be solved numerically to obtain the optimal function $p(t)$.

7 Productivity

In this section, we apply the results from Section 6 to calculate productivity. We define productivity as the value of a product divided by the cost to produce the product, which might also be called the return on investment. For the linear work model, as Figure 4 shows, the work is $k^* = W_*/a$, from equation (6.15), and the resources used are $p^* = k^* - 1$, from equation (6.16). Productivity is, therefore, the ratio,

$$\Psi^* = \frac{1}{1 - 1/k^*} , \quad (7.1)$$

which is greater than one,

$$\Psi^* > 1 . \tag{7.2}$$

For example, $\Psi^* = 1.25$ for the result shown in Figure 4. Productivity approaches one as the value of k^* increases because resources are used at maximum power in the limit of large time. This specific behavior holds for the system defined by the linear work function of equation (6.3). The behavior of the productivity function for the linear model is not meant to imply anything about the behavior of productivity functions in general.

We can adjust the productivity function to reflect the value of the work relative to the value of the resources by multiplying by accounting charges. If V is the value of the product per unit of work, and if C is the cost for one unit of resource, then productivity is the dimensionless ratio

$$\Psi = \left(\frac{V}{C} \right) \cdot \Psi^* . \tag{7.3}$$

The ratio of accounting charges is somewhat subjective by its very nature. That being the case, we can make the value of this second definition for productivity anything we want.

8 Summary

We have outlined a new approach to performance metrics by focusing on the evolution of work as a function of time. We have argued that measuring time alone is insufficient for understanding why a project performs as it does. We have asserted that we need to observe work as it evolves in time and to examine computational action as a measure of how well a project uses the resources available to it.. We have proposed a new metric that measures the lost, or wasted, action and we have proposed that we should minimize this loss. We have called this principle a principle of computational least action.

To apply this principle, we need a model for how work evolves in time and we need a constraint that tells us when the work is done. We can then measure action as the integral of work over time and to compare it with the action that could be generated at full power. We want to minimize the difference between to two.

If the work model depends on a set of resources that also evolve in time, we have shown how to apply the classical techniques of the calculus of variations to find the optimal way to apply those resources. In a real project, these methods lead to a very complicated system of differential equations along with a very complicated set of constraints. It may be very difficult to apply this method, but it indicates a methodology for a new why of thinking about performance analysis.

The principle of least action is fundamental to many areas of physics [6]. Perhaps it can guide us to fundamental insights in the area of computer performance analysis as well.

References

- [1] B. N. Taylor, editor. The international system of units (SI). Technical Report Special Publication 330, National Institute of Standards and Technology, U.S. Government Printing Office, Washington, DC, 1991.
- [2] Raymond T. Birge. On the establishment of fundamental and derived units, with special reference to electric units. Part I. *American Physics Teacher*, 3:102–109, 1935.

- [3] P. W. Bridgman. *Dimensional Analysis*. Yale University Press, New Haven, 2nd edition, 1931.
- [4] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, John Wiley, 1953.
- [5] James J. Hack. Peak vs. sustained performance in highly concurrent vector machines. *IEEE Computer*, pages 11–19, September 1986.
- [6] Cornelius Lanczos. *The Variational Principles of Mechanics*. Dover Publications, Inc., New York, 4th edition, 1986.
- [7] Robert W. Numrich. The computational action norm and the principle of computational least action. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, March 1997. SIAM Activity Group on Supercomputing, Society for Industrial and Applied Mathematics.
- [8] Robert W. Numrich. Computational force, mass and energy. *International Journal of Modern Physics C*, 8(3):437–457, 1997.
- [9] National Institute of Standards and Technology. Nist reference on constants, units, and uncertainty: Prefixes for binary multiples, 1998, <http://physics.nist.gov/cuu/Units/binary.html>.
- [10] TOP500 Supercomputer Sites. <http://www.top500.org>, June 2003.
- [11] Akhilesh Tyagi. A principle of least computational action. In *Proc. PhysComp '92: Workshop on Physics and Computation*, pages 262–266. IEEE, October 1992.
- [12] Adrian T. Wong, Leonid Oliker, William T.C. Kramer, Teresa L. Kaltz, and David H. Bailey. ESP: A system utilization benchmark. In *Proceedings of Supercomputing 2000*, November 2000.

New Rank	η (Zflop·s)	Old Rank
1	0.03	40
2	0.04	33
3	0.04	28
4	0.10	29
5	0.19	112
6	0.27	10
7	0.29	9
8	0.39	3
9	0.41	38
10	0.48	32
11	0.55	25
12	0.62	36
13	0.68	31
14	0.76	20
15	0.76	6
16	0.80	11
17	0.83	26
18	0.83	27
19	0.86	24
20	0.90	19
21	0.98	12
22	0.99	18
23	1.13	1
24	1.15	44
25	1.47	2
26	1.64	8
27	2.35	5
28	2.39	13
29	2.46	34
30	2.69	17
31	3.62	30
32	3.81	4
33	3.93	21
34	8.69	7

Table 3: The computational action metric for selected systems from the Top500 list.