Perfect strong scaling using no additional energy

James Demmel
Andrew Gearhart
Oded Schwartz
Benjamin Lipshitz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-126
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-126.html

May 30, 2012

Perfect strong scaling using no additional energy

James Demmel [*]
UC Berkeley
demmel@cs.berkeley.edu

Andrew Gearhart [*]
UC Berkeley
agearh@cs.berkeley.edu

Benjamin Lipshitz [*]
UC Berkeley
lipshitz@berkeley.edu

Oded Schwartz [†]
UC Berkeley
odedsc@cs.berkeley.edu

Abstract

Energy efficiency of computing devices has become a dominant area of research interest in recent years. Most of this work is focused on architectural techniques to improve power and energy efficiency; only a few consider saving energy at the algorithmic level. We prove that a region of *perfect strong scaling in energy* exists for matrix multiplication (classical and Strassen) and the direct ($O(n^2)$) n-body problem via the use of .5D algorithms: This means that we can increase the number of processors by a constant factor, with the runtime (both computation and communication) decreasing by the same factor, and the total energy used remaining constant.

Keywords: Energy lower bounds, Communication-avoiding algorithms, .5D algorithms

[*]Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung.

[†]Research supported by U.S. Department of Energy grants under Grant Number DE-SC0003959.

1 Introduction and Motivation

In recent years, energy efficiency of computing devices has become a dominant area of research interest. While a large body of work has focused upon architectural techniques to improve power and energy efficiency (see [16] for a survey through 2008), few publications consider the energy efficiency at the algorithmic level. In this work, we model algorithm runtime T and execution energy E via a small set of architectural parameters and extend previous work on communication-avoidance to derive lower bounds on the amount of energy that must be consumed during runtime. From these bounds, we prove that a realm of perfect strong scaling in energy (i.e. for a given problem size n, the energy consumption remains constant as the number of processors p increases and the runtime decreases proportionally to p) exists for matrix multiplication (classical and Strassen) and the direct ($O(n^2)$) n-body problem. In addition to these results, the bounds on energy allow us to discuss a number of problems:

1. What is the minimum energy required for a computation?
2. Given a maximum allowed runtime T, what is the minimum energy E needed to achieve it?
3. Given a maximum energy budget E, what is the minimum runtime T that we can attain?
4. The ratio $P = E/T$ gives us the average power required to run the algorithm. Can we minimize the average power consumed?
5. Given a bound on average power, can we minimize energy or runtime?
6. Given an algorithm, problem size, number of processors and target energy efficiency (GFLOPS/W), can we determine a set of architectural parameters to describe a conforming computer architecture?

To conclude, we apply the energy model to the physical parameters of an actual machine and evaluate accuracy. We scale the model parameters in an attempt to gain insight into the future technology trends required to achieve a desired level of energy efficiency.

2 Deriving lower bounds on algorithm energy consumption

Timing Model

In order to obtain bounds on energy, we first represent the runtime of an algorithm by adding the time for computation and communication on a given processor. This assumes no overlap; overlap could reduce the time by at most a factor of 2 or 3, a constant factor omitted for simplicity[1]. The time to send one message consisting of k words from one processor to another is modeled as $\alpha_t + k\beta_t$, where α_t is the latency (seconds per message), β_t is the reciprocal bandwidth (seconds per word), and $k \leq m$, where m is the maximum size of a message. We also assume $m \leq M$, where M is the (maximum) memory used. The total runtime T of a processor is then

$$T = \gamma_t F + \beta_t W + \alpha_t S \tag{1}$$

where γ_t is the seconds per flop, F is the number of flops, W is the total number of words sent, and S is the total number of messages sent.

Energy Model

To model the total energy cost E of executing an algorithm, we sum the energy costs of computation (proportional to the number of flops F), communication (proportional to the number of words W and messages S sent), memory (proportional to the memory used M times the runtime T) and "leakage" (proportional to runtime T) for each processor and multiply by the number of processors p. This results in the expression

$$E = p(\gamma_e F + \beta_e W + \alpha_e S + \delta_e M T + \epsilon_e T) \tag{2}$$

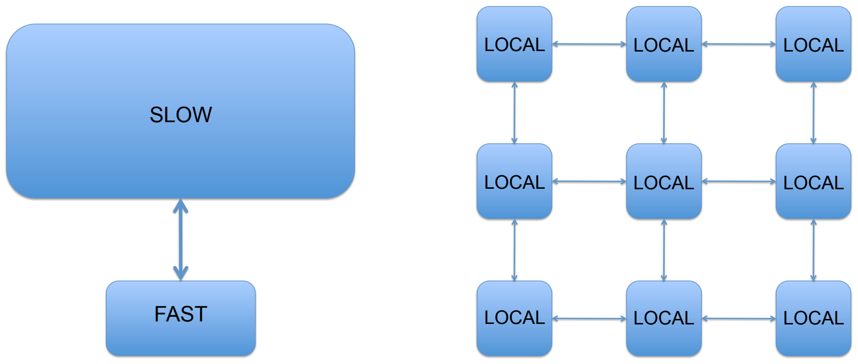
Here γ_e, β_e and α_e are the energy costs (in joules) per flop, per word transferred and per message, respectively. δ_e is the energy cost per stored word per second. The term $\delta_e MT$ assumes that we only pay for energy on memory that we are utilizing for the duration of the algorithm (a strong architectural assumption). ϵ_e is the energy leakage per second in the system outside the memory. Note that ϵ_e encompasses the static leakage energy from circuits as well as the energy of other devices not defined within the model such as disk behavior or fan activity.

[1]The model is flexible, and overlapping could be represented by a max operation over the runtime components.

Machine Model

In this work and as in [7], we consider the same abstract model of a distributed machine shown in Figure 1(b). In this model, each processing node is homogeneous (i.e. identical γ and M parameters) and linked within an abstract network topology. To communicate, words are packed into contiguous messages before being communicated to another processor. Furthermore, as the number of processors within the distributed machine scales we assume that the link parameters (α and β parameters) remain constant (more on this to be discussed later). The link transmissions define communication between the local memories of two individual processing elements. This model can be applied to any physical machine that involves a communication network; i.e. large clusters or System on Chip (SoC) designs with an on-board communication network between processing cores.

We note that the algebraic model derived here for the distributed machine can be extended or modified to suit the desired future machine environment with greater accuracy. This can aid in utilizing the lower bounds on algorithm characteristics to aid hardware development. As an example, in the cases of the 1.5D n-body problem and 2.5D matrix multiplication we also will present energy models for a machine of the type presented in Figure 2.



(a) Two-level sequential model (b) Distributed parallel model

Figure 1: Abstract machine models

This more complicated model will not be analyzed further in this work, but allows for a greater degree of parameterization by modeling two levels of machine communication (as opposed to the one-level distributed machine model primarily discussed).

3 Background on Communication Avoiding Algorithms

Communication lower and upper bounds

If one considers the execution of an algorithm as a combination of computational and communication operations, it is natural to attempt to derive lower bounds on the amount of communication required to compute a given problem. In [7], we extend the work of Hong and Kung [15] and Irony, Toledo and Tiskin [14] to prove a general lower bound on the amount of data moved (i.e., bandwidth-cost) by a general class of linear algebra algorithms. This result holds for most of direct linear algebra algorithms including Basic Linear Algebra Subroutine (BLAS) [8] operations (e.g. matrix-

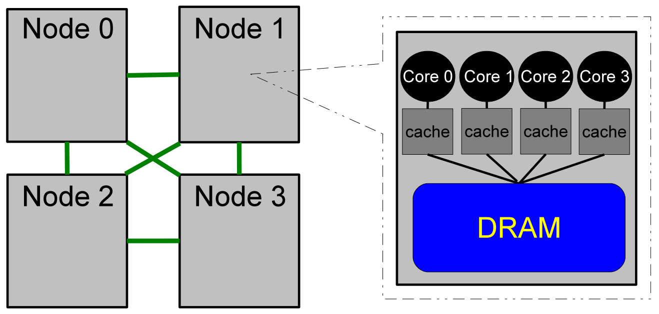


Figure 2: Two level machine model with 4 nodes and 4 cores per node

vector multiplication, matrix multiplication, and triangular solve with one or multiple right hand sides) and computing LU, Cholesky, LDL^T, and QR decompositions, as well as many eigenvalue/SVD computations. The result holds whether the matrices are dense or sparse, and whether the machine fits a two-level (Figure 1(a)) or distributed memory model (Figure 1(b)).

In the sequential model, if a processor does F floating point operations (flops) that satisfy the requirements described in [7] and utilizes M words of fast memory, then the total number of words W sent and received by the processor satisfies

$$W \geq \max\left(I + O, \frac{F}{M^{\frac{1}{2}}}\right). \tag{3}$$

where I and O are the number of input and output words, respectively[2]. Following [7], we obtain a lower bound on the number of messages S a processor sends and receives by dividing the lower bound on the number of words given in (5) by the size of the largest possible message m. Thus, if a processor executes F flops as before, we have

$$S \geq \max\left(\frac{I + O}{m}, \frac{F}{mM^{\frac{1}{2}}}\right). \tag{4}$$

A similar expression to Equation 5 bounds word traffic in the parallel model

$$W \geq \max\left(0, \frac{F}{M^{\frac{1}{2}}} - (I + O)\right). \tag{5}$$

with the parallel message bound derived in a similar manner to Equation 4. In the parallel situation, if the $I + O$ term is larger than the amount of data needed to do the flops, it is conceivable that there exists a parallel algorithm with no communication assuming the correct data layout.

In the case of matrix-matrix operations (BLAS3 functions), we have $F = O(n^3)$ and $I + O = O(n^2)$, so the second term of the above bounds usually dominates. On the other hand, for matrix-vector and vector-vector operations (BLAS2 and BLAS1 functions, respectively) the size of the input and output data is the maximal term. These lower bound results have been utilized to prove the communication optimality of several new linear algebra algorithms [5], and have also been extended to a model of heterogeneous processing [2].

2.5D algorithms

In the case of parallel matrix multiplication where data is distributed in blocks on a $p^{\frac{1}{2}}$-by-$p^{\frac{1}{2}}$ grid, each processor performs $F = O(n^3/p)$ flops and utilizes $M = \Omega(n^2/p)$ words of memory. This is the situation when considering well-known methods such as Cannon's algorithm [9] or SUMMA [13]. For reasons that will soon become clear, we refer to these algorithms as "2D". In Agarwal et al. [1], a matrix multiplication algorithm for utilizing redundant copies of the input matrices is presented. Here, the input data is distributed on a $p^{\frac{1}{3}}$-by-$p^{\frac{1}{3}}$-by-$p^{\frac{1}{3}}$ cube of processors and we hereafter refer to this algorithm as a "3D" matrix multiplication algorithm. In 3D matrix multiplication, the amount of local data increases to $M = \Theta(n^2/p^{\frac{2}{3}})$ and results in a factor $p^{\frac{1}{6}}$ reduction in words communicated, and an even larger savings in the number of messages (see [18] for more details).

In [18], Solomonik and Demmel propose an algorithm for matrix multiplication that utilizes redundant copies of input matrices in a range between that of the 2D and 3D algorithms. In other words, the "2.5D" matrix multiplication algorithm can use any amount of local memory in a range

$$\frac{n^2}{p} \leq M \leq \frac{n^2}{p^{2/3}} \tag{6}$$

and distributes data on a $(p/c)^{\frac{1}{2}}$-by-$(p/c)^{\frac{1}{2}}$-by-c cuboid of processors where c is a data replication factor. This algorithm has optimal communication costs of

$$W = O\left(\frac{n^2}{(cp)^{\frac{1}{2}}}\right), S = O\left(\left(\frac{p}{c^3}\right)^{\frac{1}{2}} + log\ c\right) \tag{7}$$

Note that when $c = 1$ ($M = n^2/p$), the algorithm reduces to the classical 2D algorithm and when $c = p^{\frac{1}{3}}$ ($M = n^2/p^{\frac{2}{3}}$) it reduces to 3D matrix multiplication.

[2]If the original input data does not reside in fast memory at the start of the algorithm and the final output data must be written out of fast memory at the end of the algorithm, then there is a trivial lower bound based on the sum $I + O$ of input and output words.

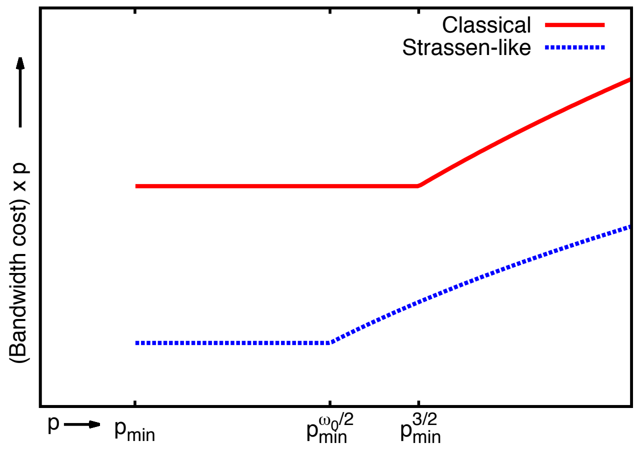


Figure 3: Limits of communication strong scaling for matrix multiplication [4, 6]

A key observation regarding 2.5D matrix multiplication is that the algorithm achieves perfect strong scaling (i.e. for the same problem size and twice the processors, we can halve the algorithm's runtime) within the range of $n^2/p \leq M \leq n^2/p^{\frac{2}{3}}$. To see this, we consider p_{min} to be the smallest number of processors that can fit a given problem size. In this situation, we must use a 2D algorithm ($M = \Theta(n^2/p_{min})$) as we are unable replicate the input data to reduce communication. Thus, $W_{p_{min}} = O(n^2/p_{min}^{\frac{1}{2}})$ and $S_{p_{min}} = O(p_{min}^{\frac{1}{2}})$. If we scale the problem to $p = cp_{min}$ processors, we have $M = \Theta(cn^2/p_{min})$ and can utilize the 2.5D algorithm with a bandwidth cost of $O(n^2/(c^2p_{min})^{\frac{1}{2}}) = W_{p_{min}}/c$ and message cost of $O((p_{min}/c^2)^{\frac{1}{2}}) = S_{p_{min}}/c$. Thus, by utilizing c more processors to solve the problem we can create redundant copies of the input data to keep the communication volume constant [3]. Solomonik and Demmel [18] also propose an algorithm for 2.5D LU factorization that is bandwidth optimal ($W_{2.5DLU} = O(n^2/(cp)^{\frac{1}{2}})$ but requires a larger $S_{2.5DLU} = \Omega((cp)^{\frac{1}{2}})$ messages (which attains a different lower bound that applies to LU but not matrix multiplication).

Unfortunately, as proven by Ballard et al. in [4], the tactic of utilizing more memory to preserve strong scaling properties does not work indefinitely. In fact, this perfect strong scaling for matrix multiplication cannot continue past $p = \Omega(n^3/M^{\frac{3}{2}})$ (or $p = \Omega(n^{w_o}/M^{\frac{w_0}{2}})$ for fast matrix multiplication algorithms that multiply two $n \times n$ matrices in $\Theta(n^{w_0})$ time, like Strassen), where there is no way to use more memory to reduce communication cost. These trends can be seen in Figure 3, where communication costs scale at a rate of $1/P^{\frac{2}{3}}$ and $1/P^{\frac{2}{w_0}}$ once the ability to utilize additional memory has saturated.

4 Time and energy lower bounds for various algorithms

Classical Matrix Multiplication ($O(n^3)$ algorithm)

In the case of linear algebra algorithms (including matrix-matrix multiplication) that perform $O(n^3)$ flops, we know the following expressions for F, W and S in Equation 1 from the results in [7]:

$$F = \frac{n^3}{p}, \quad W = \frac{n^3}{pM^{\frac{1}{2}}}, \quad S = \frac{W}{m} \tag{8}$$

As before, M is the memory used per processor (which cannot exceed the physical memory per processor), m is the size of the largest message we can send ($m \leq M$), and p is the number of processors. We assume that we use at least enough memory to store one copy of the data across all the processors, so $M \geq n^2/p$ (we again omit constant factors for simplicity).

From prior work on 2.5D matrix multiplication [18], we know that we can utilize redundant copies of matrices (increase M) to decrease the amount of required communication (i.e. decrease W and S). In standard "2D" algorithms for matrix multiplication, each processor is given a local of problem of size $M = \Theta(n^2/p)$ on which to work, i.e. one copy of the data is evenly spread across the processors.

[3]For this to be true, the $log\ c$ term in the expression for S in Equation 7 does not dominate.

If equations (1), (2) and (8) are combined, we obtain the following attainable lower bounds on the amount of time and energy required to run $O(n^3)$ parallel matrix multiplication[4]:

$$T_{2.5DMM}(n,p,M) = \frac{\gamma_t n^3}{p} + \frac{\beta_t n^3}{M^{1/2} p} + \frac{\alpha_t n^3}{m M^{1/2} p} \tag{9}$$

$$\begin{aligned} E_{2.5DMM}(n,p,c) &= (\gamma_e + \gamma_t \epsilon_e) n^3 + \left((\beta_e + \beta_t \epsilon_e) + \frac{(\alpha_e + \alpha_t \epsilon_e)}{m} \right) \frac{n^3}{M^{\frac{1}{2}}} \\ &\quad + \delta_e \gamma_t M n^3 + \left(\delta_e \beta_t + \frac{\delta_e \alpha_t}{m} \right) M^{\frac{1}{2}} n^3 \end{aligned} \tag{10}$$

In our above discussion of 2.5D matrix multiplication, we observed that for $p_{min} = n^2/M \le p \le n^3/M^{\frac{3}{2}}$, communication costs scale perfectly with increasing p. Thus, each term of the runtime expression $T_{2.5DMM}$ decreases proportionately to p in this range. Because each term of the energy expression $E_{2.5DMM}$ is proportional to some term of $T_{2.5DMM}(n,p,M)$, the energy stays constant as we increase the number of processors with a constant amount of memory per processor. At the 3D limit where $p = n^3/M^{\frac{3}{2}}$, the total energy is

$$\begin{aligned} E_{3DMM}(n,p) &= (\gamma_e + \gamma_t \epsilon_e) n^3 + \left((\beta_e + \beta_t \epsilon_e) + \frac{(\alpha_e + \alpha_t \epsilon_e)}{m} \right) n^2 p^{1/3} \\ &\quad + \delta_e \gamma_t n^5 \frac{1}{p^{2/3}} + \left(\delta_e \beta_t + \frac{\delta_e \alpha_t}{m} \right) n^4 \frac{1}{p^{1/3}} \end{aligned} \tag{11}$$

Increasing p in the 3D case reduces the energy costs due to memory usage, but increases the energy costs due to communication.

It is reasonable to ask whether our model of constant communication costs in time (β_t and α_t) and energy (β_e and α_e) makes sense as p grows, since this makes implicit assumptions about the interconnection network. Our prior work shows that a 3D torus network is a perfect match to this algorithm [17], and scales in total size proportionally to p, so the $p\epsilon_e T$ term in E should capture its energy usage. As a side note, if we consider the two level machine model of Figure 2 we obtain these expressions for runtime and energy

$$T = \frac{\gamma_t n^2}{p} + \frac{\beta_t^n n^3}{p_n \sqrt{M_n}} + \frac{\beta_t^\ell n^3}{p \sqrt{M_\ell}}$$

$$\begin{aligned} E = n^3 \Bigg[& \gamma_e + \gamma_t \epsilon_e + \frac{\beta_e^n + \beta_t^n \epsilon_e p_\ell}{\sqrt{M_n}} + \frac{\beta_e^\ell + \beta_t^\ell \epsilon_e}{\sqrt{M_\ell}} + \gamma_t \left(\delta_e^n \frac{M_n}{p_\ell} + \delta_e^\ell M_\ell \right) + \\ & \left(\delta_e^n \frac{M_n}{p_\ell} + \delta_e^\ell M_\ell \right) \left(\frac{\beta_t^n p_\ell}{\sqrt{M_n}} + \frac{\beta_t^\ell}{\sqrt{M_\ell}} \right) \Bigg] \end{aligned}$$

where p_n,β^n,α^n,M_n and δ^n are the number of nodes, internode link word cost, internode message cost, node memory size and node memory storage cost, respectively. Similar parameters for the intra-node characteristics are defined with a superscript l [5] and $p = p_n p_l$. In the above expressions, the latency portion of the communication has been eliminated for simplicity. It can be added by substituting $\beta = \beta m + \alpha$ (with the appropriate m_n or m_l).

Strassen's Matrix Multiplication

Fast matrix multiplication algorithms multiply two $n \times n$ matrices in $\Theta(n^{\omega_0})$ time, for some $2 < \omega_0 < 3$. For example, Strassen's algorithm has exponent $\omega_0 = \log_2 7 \approx 2.81$. Using the Communication-Avoiding Parallel Strassen (CAPS) algorithm [3], it is possible to perform fast matrix multiplication with less communication than classical matrix multiplication. Repeating the analysis from above, we find that the total energy is:

$$\begin{aligned} E_{FLM}(n,p,M) &= (\gamma_e + \gamma_t \epsilon_e) n^{\omega_0} + \left((\beta_e + \beta_t \epsilon_e) + \frac{(\alpha_e + \alpha_t \epsilon_e)}{m} \right) \frac{n^{\omega_0}}{M^{\omega_0/2-1}} \\ &\quad + \delta_e \gamma_t M n^{\omega_0} + \left(\delta_e \beta_t + \frac{\delta_e \alpha_t}{m} \right) M^{2-\omega_0/2} n_0^{\omega} \end{aligned} \tag{12}$$

[4]Note that choosing M equal to its maximum value $n^2/p^{2/3}$ causes a $\log p$ term to appear in the latency component. We omit this for simplicity.
[5]With the exception of p_l and M_l which represent the number of cores per node and size of core local memory.

in the case that $n^2/p \le M \le n^2/p^{2/\omega_0}$ (here FLM means "Fast matrix multiplication using Limited Memory."). When $M = n^2/p^{2/\omega_0}$

$$E_{FUM}(n,p) = (\gamma_e + \gamma_t\epsilon_e)n^{\omega_0} + \left((\beta_e + \beta_t\epsilon_e) + \frac{(\alpha_e + \alpha_t\epsilon_e)}{m}\right) n^2 p^{1-2/\omega_0} + \delta_e\gamma_t n^5 \frac{1}{p^{2/\omega_0}} + \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right) n^4 \frac{1}{p^{4/\omega_0 - 1}}, \qquad (13)$$

where FUM means "Fast matrix multiplication using Unlimited Memory." As in the case of classical matrix multiplication, the energy does not depend on p inside a perfect strong scaling range, so scaling p by some factor while holding M constant reduces the execution time by that factor without affecting the total energy. We can use this model to answer the same optimization questions as in the classical case, but analytic solutions are harder to obtain because ω_0 appears in the powers of M.

LU factorization

For dense LU decomposition, the optimal algorithm does strongly scale in the bandwidth term (i.e. it is identical, modulo constant factors, to the $\beta_t n^3/(pM^{1/2})$ term in 2.5D matrix multiplication). However, it does not scale in the latency term: $\alpha_t n M^{\frac{1}{2}}/m$. Whether this latency term is important depends on the machine constants. This is an interesting area of exploration, as the structure of LU has a critical path very similar to that of many other linear algebra problems.

Direct n-body problem

Another example where perfect strong scaling is possible is the direct ($O(n^2)$) implementation of the n-body algorithm, where each particle (or "object") has to directly interact with every other particle (this is not limited to gravity or electrostatics, any interaction where we can basically just "sum" the results of individual interactions works). In Figure 4, one can observe several possible variants of this algorithm. Like the cases of 2D and 3D algorithms, we can replicate data upon processors to reduce the amount of required communication. In the "1D" version of the direct n-body problem there is no replication and we move n words. In the "2D" version, we map the processors onto a $p^{\frac{1}{2}}$-by-$p^{\frac{1}{2}}$ grid and replicate the input data $\sqrt{p}$ times to reduce the number of words moved by $\sqrt{p}$. Based on these descriptions, one can imagine a "1.5D" variant of the direct n-body problem that utilizes memory in the range $n/p \le M \le n/p^{\frac{1}{2}}$. More details can be found in [12]. To address the number of flops performed in the algorithm, we add an additional parameter f that represents the number of flops necessary to compute the interaction of a pair of particles; although f is a constant, it may be quite large. The costs are

$$F = \frac{fn^2}{p}, \quad W = \frac{n^2}{pM}, \quad S = \frac{W}{m}$$

in the range $n/p \le M \le n/p^{\frac{1}{2}}$. Thus

$$T_{1.5D}(n,p,M) = \frac{\gamma_t f n^2}{p} + \frac{\beta_t n^2}{Mp} + \frac{\alpha_t n^2}{mMp}, \qquad (14)$$

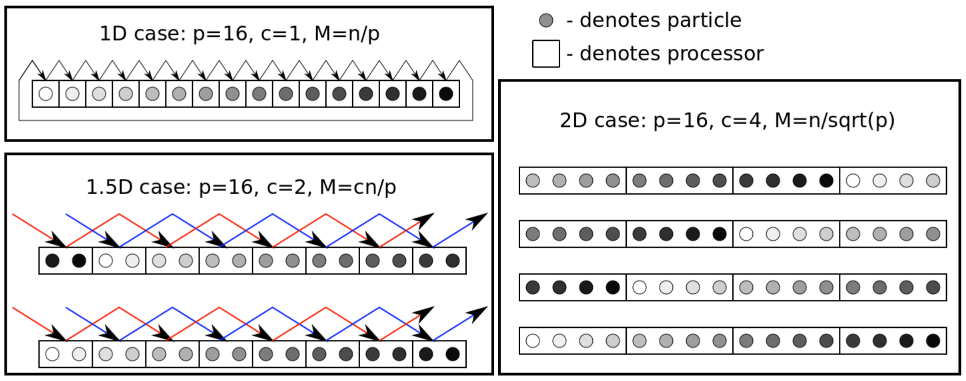
$$E_{1.5D}(n,M) = (f(\gamma_e + \gamma_t\epsilon_e) + \delta_e(\beta_t + \alpha_t/m))n^2 + \left((\beta_e + \beta_t\epsilon_e) + \frac{\alpha_e + \alpha_t\epsilon_e}{m}\right)\frac{n^2}{M} + \delta_e\gamma_t f M n^2. \qquad (15)$$

Via a similar argument to that of matrix multiplication, we can see that the 1.5D n-body algorithm achieves perfect energy scaling within the range of $n/p \le M \le n/p^{\frac{1}{2}}$. Again, we also note the expressions for a two level model of the form presented in Figure 2:

$$T = \frac{fn^2\gamma_t}{p} + \frac{\beta_t^n n^2}{M_n p_n} + \frac{\beta_t^\ell n^2}{M_\ell p}$$

$$E = n^2 \Big[(f\gamma_e + f\gamma_t\epsilon_e + \delta_e^n\beta_t^n + \delta_e^\ell\beta_t^\ell) + (p_\ell\beta_e^n + \epsilon_e p_\ell\beta_t^n)M_n^{-1} + (\beta_e^\ell + \epsilon_e\beta_t^\ell)M_\ell^{-1} + \frac{\delta_e^\ell}{p_\ell} f\gamma_t M_n + \delta_e f\gamma_t M_\ell + \frac{\delta_e^n\beta_t^\ell M_n}{p_\ell M_\ell} + \frac{\delta_e p_\ell \beta_t^n M_\ell}{M_n} \Big]$$

The parameters in these expressions are defined as in the two level equations for matrix multiplication, above. As before, the latency component to the equations can be added by substituting $\beta = \beta m + \alpha$.



1D case: p=16, c=1, M=n/p
1.5D case: p=16, c=2, M=cn/p
2D case: p=16, c=4, M=n/sqrt(p)
- denotes particle
- denotes processor

Figure 4: Data layouts for 1D, 1.5D and 2D N-body algorithms

Fast Fourier Transform (FFT)

A Fast Fourier Transform (FFT) with input size n performs $n \log n$ flops in $\log n$ steps. The usual way to parallelize it is to divide the data between the p processors in a cyclic fashion, so that the first $\log(n/p)$ steps can be performed without communication. Then an all-to-all communication of all the data is needed, after which this pattern repeats. Note that if $p \leq n^c$ for some constant $c < 1$, then only a constant number of communication steps are needed. Using a naive implementation of the all-to-all, the costs are

$$F = \frac{n \log n}{p}, \quad W = \frac{n}{p}, \quad S = p.$$

Alternately, the message count can be reduced at the cost of a higher word count, using a tree-based all-to-all, giving

$$F = \frac{n \log n}{p}, \quad W = \frac{n \log p}{p}, \quad S = \log p.$$

In either case, there is no perfect strong scaling range, since the message count does not scale. Additionally, there is no use for extra memory, so we always take $M = n/p$. Using the second choice of communication costs, the time is

$$T_{FFT}(n,p) = \frac{\gamma_t n \log n}{p} + \frac{\beta_t n \log p}{p} + \alpha_t \log p,$$

and the energy is

$$E_{FFT} = (\gamma_e + \epsilon_e \gamma_t) n \log n + (\beta_e + \epsilon_e \beta_t + \delta_e \alpha_t) n \log p + (\alpha_e + \epsilon_e \alpha_t) p \log p + \frac{\delta_e \gamma_t n^2 \log n}{p} + \frac{\delta_e \beta_t n^2 \log p}{p}.$$

Because of the $\log p$ factors, we won't be able to optimize these in closed form.

5 Applications of energy bounds to 2.5D Matrix Multiplication

We mentioned several of these in the beginning of this document:

Minimizing energy for the computation. Note that the energy usage in Equation (10) is independent of p; we want to choose the value of M that optimizes energy use. Setting

$$B = (\beta_e + \beta_t \epsilon_e) + \frac{(\alpha_e + \alpha_t \epsilon_e)}{m}, \qquad C = \delta_e \gamma_t, \qquad D = \delta_e \beta_t + \frac{\delta_e \alpha_t}{m},$$

we want to solve

$$\frac{dE}{dM^{\frac{1}{2}}} = n^3 \left(-\frac{B}{M} + 2CM^{\frac{1}{2}} + D \right) = 0,$$

which has a unique positive solution

$$F = \left(6\sqrt{3}(27B^2C^4 - BC^2D^3)^{\frac{1}{2}} + 54BC^2 - D^3 \right)^{\frac{1}{3}}, \qquad M_0 = \left(\frac{1}{6} \left(\frac{F}{C} + \frac{D^2}{CF} - \frac{D}{C} \right) \right)^2.$$

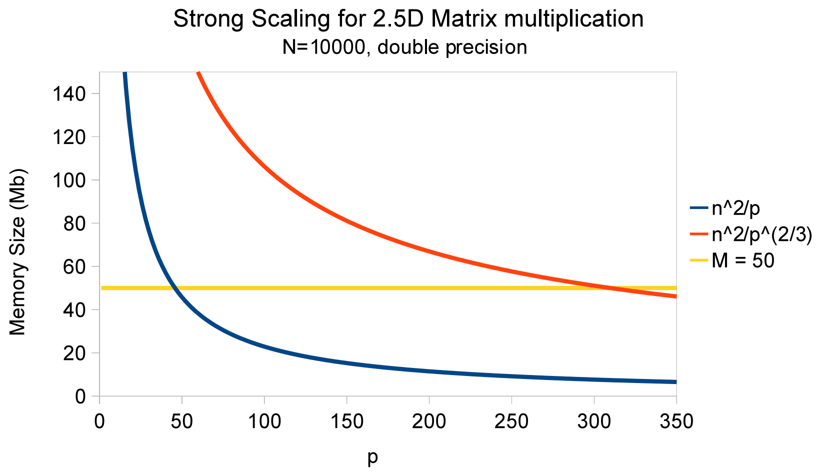


Figure 5: Strong scaling region for situation where M_0 = 50Mb

As noted above, the energy use is the same in the range $n^2/p \le M \le n^2/p^{2/3}$, and M is held fixed. This means it is possible to attain the minimum energy use for any

$$\frac{n^2}{M_0} \le p \le \frac{n^3}{M_0^{3/2}}.$$

In Figure 5, we illustrate the region of perfect energy strong scaling for M_0 = 50Mb (and N=10000). In this situation, we can theoretically expect perfect scaling between about 50 and 310 processing cores (the region along M=50 between the bounds of n^2/p and $n^2/p^{2/3}$).

Minimizing energy given an upper bound on the run time. Again this can be solevd in closed form. There are two cases.

1. It is possible to achieve the required runtime T_{max} and the minimum energy use. This is the case if $T_{2.5DMM}(n, n^3/M_0^{3/2}, M_0) \le T_{max}$. In this case, for example, taking $p = n^3/M_0^{3/2}$ and $M = M_0$ is optimal.
2. Otherwise, we need to use the 3D algorithm to achieve the desired runtime. In this case, we must use at least p_{min} processors, where $T_{3DMM}(n, p_{min}) = T_{2.5DMM}(n, p_{min}, n^2/p_{min}^{2/3}) = T_{max}$. Finally, one chooses the value of p that minimizes $E_{3dMM}(n, p)$, subject to the constraint that $p \ge p_{min}$.

Minimizing runtime given an upper bound on energy. Conversely, if we fix the maximum allowed energy, this will limit M to lie in some range. Minimizing the runtime subject to this constraint will hit the boundary $M = n^2/p^{2/3}$ of the feasible region, i.e. we would use the 3D matrix multiplication algorithm, and p is taken as the largest value such that $E_{3DMM}(n, p) = E_{max}$.

Minimizing average power By considering that average power for a processor is $p = \frac{E_p}{T}$ (and assuming that all processors complete the task utilizing the same amount of time and energy), we can combine our previous expressions for E_p and T to obtain an expression for power P:

$$P = \frac{E}{T} = \frac{\frac{\gamma_e n^3}{p} + \frac{\beta_e n^3}{pM^{\frac{1}{2}}} + \frac{\alpha_e n^3}{pM^{\frac{1}{2}}m} + \delta_e MT + \epsilon_e T}{T}$$

$$= \frac{\left(\frac{\gamma_e n^3}{p} + \frac{\beta_e n^3}{pM^{\frac{1}{2}}} + \frac{\alpha_e n^3}{pM^{\frac{1}{2}}m}\right)}{\left(\frac{\gamma_t n^3}{p} + \frac{\beta_t n^3}{pM^{\frac{1}{2}}} + \frac{\alpha_t n^3}{pM^{\frac{1}{2}}m}\right)} + \delta_e M + \epsilon_e$$

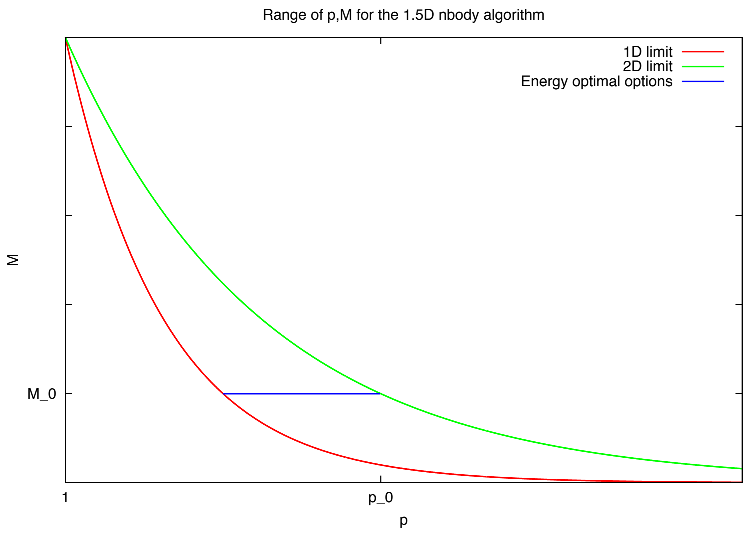


Figure 6: The 1D and 2D limits of the 1.5D nbody algorithm. The algorithm can only be run for values of p and M that lie in this range. Note that moving horizontally in this range corresponds to perfect strong scaling: decreasing the running time proportionally to p while keeping the energy use optimal. The energy optimal options are when $M = M_0$, as defined below. Among the energy optimal options, taking $p = p_0$ at the 2D limit gives the fastest execution. Faster executions can be achieved by increasing p along the 2D line, at the expense of higher energy use.

thus,

$$P = \frac{\gamma_e M^{\frac{1}{2}} m + \beta_e m + \alpha_e}{\gamma_t M^{\frac{1}{2}} m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e$$

If the goal is to minimize average power, there are two cases.

1. If
$$\frac{\beta_e m + \alpha_e}{\gamma_e} > \frac{\beta_t m + \alpha_t}{\gamma_t},$$
then there is a unique positive value of c/p that minimizes power, which is given by the square of the positive root of a quartic polynomial. Note that this inequality can be interpreted as: the number of flops that can be performed using the same *energy* as sending a maximum-length message is greater than the number of flops that can be performed in the same *time* as sending a maximum-length message.
2. Otherwise, the power is an increasing function of c/p for all positive values, and the minimum average power is attained by setting $c = 1$, and p as large as possible, and the power approaches
$$P_{2.5DMM}(n, \infty, 1) = \frac{\beta_e m + \alpha_e}{\beta_t m + \alpha_t} + \epsilon_e.$$

We can also ask questions like how to optimize the power given an upper bound T_{max} on runtime, as above. Like $E_{2.5DMM}(n, p, c)$, the power $P_{2.5DMM}(n, p, c)$ also just depends on the parameter p/c.

6 Applications of energy bounds to the direct n-body problem

In the case of the n-body algorithm, many questions can be answered with relatively simple expressions.

Minimizing energy for the computation. Total energy is minimized by using memory

$$M_0 = \left(\frac{\beta_e + \beta_t \epsilon_e + (\alpha_e + \alpha_t \epsilon_e)/m}{\delta_e \gamma_t f} \right)^{\frac{1}{2}}.$$

The minimum energy is

$$E^*_{1.5D}(n) = E_{1.5D}(n, M_0) = n^2 \left(f(\gamma_e + \gamma_t \epsilon_e) + \delta_e(\beta_t + \alpha_t/m) + 2\left(\delta_e \gamma_t f \left(\beta_e + \beta_t \epsilon_e + (\alpha_e + \alpha_t \epsilon_e)/m\right)\right)^{\frac{1}{2}} \right)$$

It is possible to attain the minimum energy use for p in the range

$$\frac{n}{M_0} \le p \le \frac{n^2}{M_0^2}.$$

Figure 6 illustrates the region of strong scaling in energy for the 1.5D algorithm and explains the behavior of the algorithm within the region of strong scaling.

Minimizing energy given an upper bound on the run time. There are two cases

1. If
$$T_{max} \ge \gamma_t f M_0^2 + (\beta_t + (\alpha_t/m))M_0$$
then it is possible to achieve the absolute minimum energy $E^*_{1.5D}(n)$ within time T_{max}, for example by setting $M = M_0, p = n^2/M_0^2$.

2. Otherwise, it is necessary to use the 2D algorithm to achieve the running-time bound. To be precise, it is necessary to use at least
$$\sqrt{p_{min}} = \frac{(\beta_t + \alpha_t/m)n + \left((\beta_t + \alpha_t/m)^2 n^2 + 4T_{max}\gamma_t f n^2\right)^{\frac{1}{2}}}{2T_{max}}$$
processors. Note that the global minimum of $E_{2D}(n,p) = E_{1.5D}(n, p, n/p^{\frac{1}{2}})$ is at
$$p_0 = \frac{\delta_e \gamma_t f n^2}{\beta_e + \beta_t \epsilon_e + (\alpha_e + \alpha_t \epsilon_e)/m} = \frac{n^2}{M_0^2}.$$
We must have $p_{min} > p_0$, otherwise case (1) would apply, so the minimum energy to run in time at most T_{max} is attained by setting
$$p = p_{min}$$

Minimizing runtime given an upper bound on energy. Conversely, suppose we fix the maximum allowed energy E_{max} and want to minimize the running time. Minimizing T will always select a 2D run, since increasing p from a 1.5D run until it hits the 2D boundary decreases T. Further, the 2D runtime is a decreasing function of p, so we only need to determine the maximum p such that the 2D algorithm fits in the energy bound. This is the maximum of the two solutions[6]

$$\sqrt{p} = \frac{E_{max} - (f(\gamma_e + \gamma_t\epsilon_e) + \delta_e(\beta_t + \alpha_t/m)n^2)}{2n(\beta_e + \beta_t\epsilon_e + (\alpha_e + \alpha_t\epsilon_e)/m)} \pm$$
$$\pm \frac{\left(\left(-E_{max} + (f(\gamma_e + \gamma_t\epsilon_e) + \delta_e(\beta_t + \alpha_t/m))n^2\right)^2 - 4(\beta_e + \beta_t\epsilon_e + (\alpha_e + \alpha_t\epsilon_e)/m)n^4\delta_e\gamma_t f\right)^{\frac{1}{2}}}{2n(\beta_e + \beta_t\epsilon_e + (\alpha_e + \alpha_t\epsilon_e)/m)}$$

Minimizing runtime or energy given a bound on power By considering that average power consumed is $P = \frac{E}{T}$, we can use our previous expressions for E and T to obtain an expression for power P:

$$P_{1.5D}(n,p,c) = p\left(\frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e\right) \quad (16)$$

An upper bound on total power thus translates into an upper bound on the number of processors:

$$p \le P^{tot}_{max}\left(\frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e\right)^{-1} \quad (17)$$

[6]In some cases there are zero real solutions when the second term is imaginary.

Using the maximum number of processors, the running time simply becomes

$$T = \frac{E_{1.5D}(n,p,M)}{P_{max}^{tot}},$$

and the problem of minimizing time or energy given total power is reduced to minimizing energy, which we have already solved, with the additional constraint (17) between p and M.

Alternately we may want to minimize the runtime given a bound on the power per processor. The bound is

$$P_{max} \geq \frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e,$$

which we may solve for M to get

$$\begin{aligned}
&\frac{\gamma_t f P_{max} - \gamma_e f - \epsilon_e \gamma_t f - \delta_e(\beta_t + \alpha_t/m)}{2\delta_e\gamma_t f} - \\
&- \frac{\left((\gamma_t f P_{max} - \gamma_e f - \epsilon_e\gamma_t f - \delta_e(\beta_t+\alpha_t/m))^2 - 4\gamma_e\gamma_t f(\beta_e + \alpha_e/m - (\beta_t+\alpha_t/m)P_{max} - \epsilon_e(\beta_t+\alpha_t/m))\right)^{\frac{1}{2}}}{2\delta_e\gamma_t f} \\
&\leq M \leq \frac{\gamma_t f P_{max} - \gamma_e f - \epsilon_e \gamma_t f - \delta_e(\beta_t + \alpha_t/m)}{2\delta_e\gamma_t f} + \\
&+ \frac{\left((\gamma_t f P_{max} - \gamma_e f - \epsilon_e\gamma_t f - \delta_e(\beta_t+\alpha_t/m))^2 - 4\gamma_e\gamma_t f(\beta_e + \alpha_e/m - (\beta_t+\alpha_t/m)P_{max} - \epsilon_e(\beta_t+\alpha_t/m))\right)^{\frac{1}{2}}}{2\delta_e\gamma_t f}
\end{aligned}$$

This is the range of M for which P_{max} is attained. To minimize T, we would as many processors as possible, which is by setting M to its lower bound and taking $p = n^2/M^2$. Alternately, to minimize total energy E, note that as a function of M, E has one local minimum, discussed above. If it is in the range allowed by P_{max}, we of course make that choice. If it is above or below the allowed range, the way to minimize energy is by taking the maximum or minimum value of M, respectively.

Fix target GFLOPS/W, determine machine parameters If we fix a target number of GFLOPS/W that we wish to achieve, that is fixing the ratio

$$\frac{fn^2}{E_{1.5D}^*} = \frac{f}{\left(f(\gamma_e + \gamma_t\epsilon_e) + \delta_e(\beta_t + \alpha_t/m) + 2\left(\delta_e\gamma_t f\left(\beta_e + \beta_t\epsilon_e + (\alpha_e + \alpha_t\epsilon_e)/m\right)\right)^{\frac{1}{2}}\right)},$$

then this gives us a constraint on the machine parameters.

7 Case Study: NUMA nodes on a dual-socket server

As an example of possible uses for the energy model, we consider the physical machine configuration as presented in Figure 7. This machine has two Intel Sandy Bridge server processors (code-name Jaketown) joined via Intel's Quick Path Interconnect (QPI) [10]. Each of these processor sockets has 4 separate channels to memory, representing two Non-Uniform Memory Access (NUMA) domains. Each memory channel has 2 8Gb DIMMs for a total of 128Gb of main memory (2 NUMA nodes*4 channels/node*2 DIMMs/channel*8Gb/DIMM). Each processor has 8 physical cores running at 3.1Ghz for a total of 16 physical cores. Parameters used to seed the model are described in Table 1. To obtain γ_e, we consider the machine's peak single-precision floating point capability divided by the Thermal Design Power (TDP) of the die. This is a perhaps overly-simplistic way to model γ_e, but has the advantage of being easily calculable and represents a worst-case energy consumption scenario. Unfortunately, such a choice of parameter does not give insight into on-die sources of efficiency as on this Jaketown chip all 8 cores and caches are categorized via a single number.

In addition to the assumptions made for γ_e, we calculate γ_t based upon the peak single precision performance of the chip. Also, we assume the the leakage term $\epsilon_e = 0$ (a large assumption that needs to be investigated further) and that the energy cost per message is zero. The energy cost per word was calculated as the time to send a message multiplied by the link power and then divided by the message length.

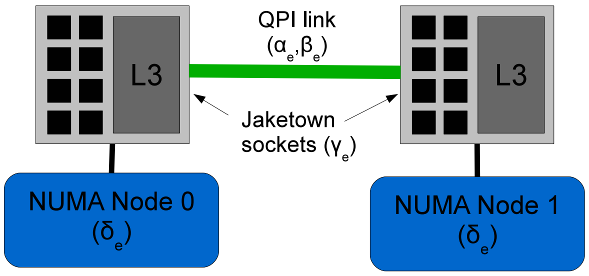


Figure 7: Dual-socket 8-core Intel Jaketown machine modeled within this case study.

Table 1: Parameters used in case study

Core Freq (Ghz)	3.1
SIMD width (Single Precision)	8
Data width (bytes)	4
Cores on Node	8
Peak FP (GFLOP/s)	396.8
M (words)	17179869184
m (words)	17179869184
Chip TDP (W)	150
Link BW (Gb/s)	25.60
Link Latency (sec)	6.000E-08
Link Active Power (W)	2.15
Link Idle Power (W)	0
DRAM DIMMS/socket	8
DRAM DIMM Power	3.1
γ_e (J/flop)	3.78024E-10
β_e (J/word)	3.78024E-10
α_e (J/msg)	0
δ_e (J/word/s)	5.7742E-09
ϵ_e (J/s)	0
γ_t (s/flop)	2.5202E-12
β_t (s/word)	1.56E-10
α_t (s/msg)	6.00E-08

To gain an initial impression of the effect from scaling γ_e, β_e and α_e, for 2.5D matrix multiplication, we hold the time parameters constant as well as the number of processors in the model (p=2, as sockets are considered processors in this case) and problem size (n=35000). Unfortunately, for such a large problem size and small number of processors we are outside the theoretical region of strong scaling in p. However, from Figure 8 we can see that doubling either γ_e, β_e or α_e independently results in a limited amount of efficiency improvement when considering the metric of GFLOPS/W. In particular, scaling β_e has almost no effect while the benefits of scaling γ_e saturate after about 5 generations (assuming parameters reduce by half with each generation). On the other hand, we obtain a desired efficiency of 75 GFLOPS/W after 5 generations if we are able to improve all three parameters together.

Of course, it is highly unlikely that energy efficiency parameters will scale without any change in the time parameters of the model. If time-dependent parameters are also scaled, the desired level of efficiency should arrive much faster. Despite the inaccurate assumptions of the model, it does show that it benefits to target energy efficiency improvements to components that benefit the system as a whole. In the above example, overall improvements can be gained by targeting on-die energy or DRAM but not the efficiency of the QPI link.

In the near future, we intend to run the 2.5D matrix multplication and 1.5D n-body codes on the actual machine modeled above to evaluate the predicted energy consumption via a wall power meter and on-chip energy counters (see [11] for more information). We also would like to obtain parameters for the SoC environment in hope of gaining insight into technology scaling within the embedded space. If we consider the problem of finding optimal machine parameters within a given energy efficiency envelope and cost metrics, we can solve the optimization problem via a steepest descents approach to guide hardware development.

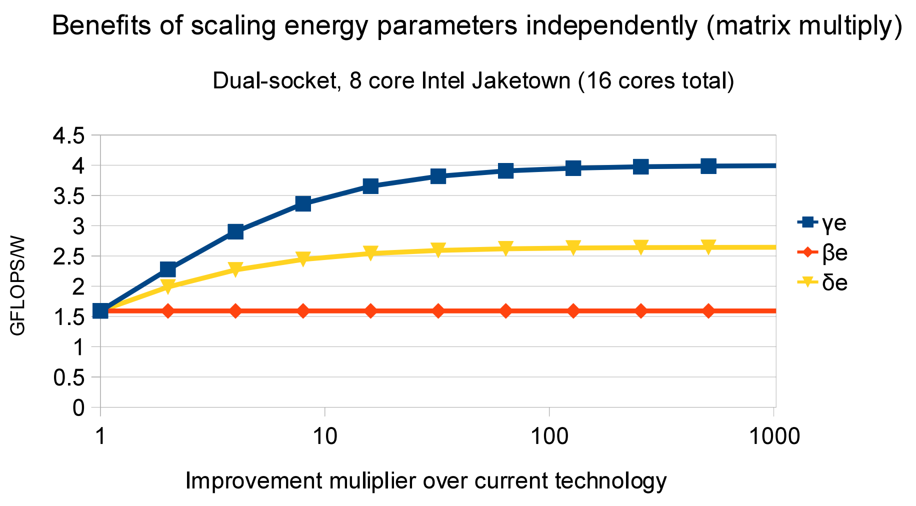


Figure 8: Scaling γ_e,β_e, δ_e independently on case study machine

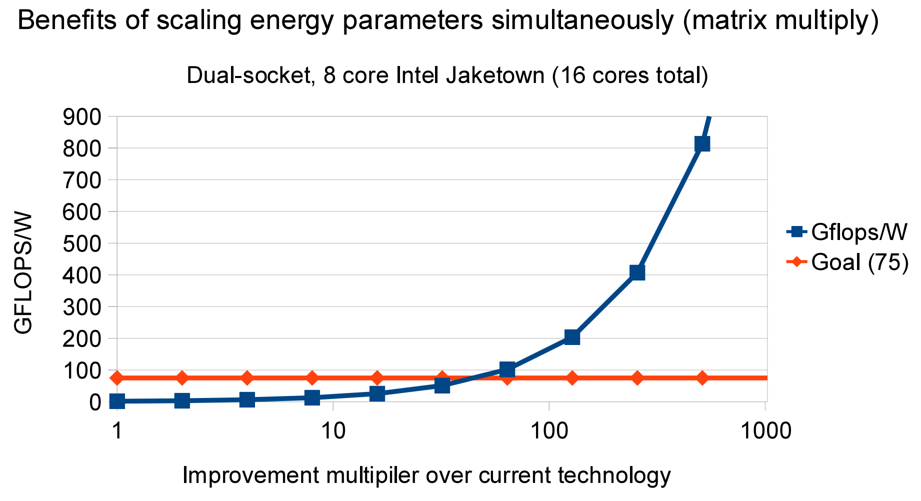


Figure 9: Scaling γ_e,β_e, δ_e together on case study machine

Table 2: Machine parameters used to determine γ_e and γ_t for various machines. SIMD width and Peak Floating Point (FP) are for single precision. TDP is Thermal Design Power.

Processor	**Core Freq (Ghz)**	**Cores**	**SIMD width**	**TDP (W)**	**Peak FP**	γ_t **(s/flop)**	γ_e **(J/flop)**	**Gflops/W**
Intel Sandy Bridge 2687W	3.1	8	8	150.0	396.80	2.52E-12	3.78E-10	2.645
Intel Ivy Bridge 3770K	3.5	4	8	77.0	224.00	4.46E-12	3.44E-10	2.909
Intel Ivy Bridge 3770T	2.5	4	8	45.0	160.00	6.25E-12	2.81E-10	3.556
Intel Westmere-EX E7-8870	2.4	10	4	130.0	192.00	5.21E-12	6.77E-10	1.477
Intel Beckton X7560	2.26	8	4	130.0	144.64	6.91E-12	8.99E-10	1.113
Intel Atom D2500	0.64	2	4	10.0	10.24	9.77E-11	9.77E-10	1.024
Intel Atom N28xx	0.64	2	4	6.5	10.24	9.77E-11	6.35E-10	1.575
Nvidia GTX480	1.401	480	1	250.0	1344.96	7.44E-13	1.86E-10	5.380
Nvidia GTX590	1.215	1024	1	365.0	2488.32	4.02E-13	1.47E-10	6.817
ARM Cortex A9	2	2	2	1.9	8.00	1.25E-10	2.38E-10	4.211
ARM Cortex A9 (low power)	0.8	2	2	0.5	3.20	3.13E-10	1.56E-10	6.400

8 Observations and Open Problems

Table 2 presents a set of data obtained for various processing devices. If the efficiency of these devices is calculated based upon peak single-precision floating point capability and TDP, we note that none are able to approach even 10 GFLOPS/W. Furthermore, the GPU examples prove to be most efficient while consuming the largest amount of power, unlike the Intel Atoms. Thus, there can be a tradeoff between peak power consumption (which needs to be bounded in some environments) and computational efficiency. This data supports the idea of power-efficient future architectures composed of large numbers of simple compute elements that allow for high parallelism without the overhead of large pipelines and speculative execution. Unfortunately, relying on parallelism for efficiency can merely push the problem into the application space without practically improving efficiency for the end-user.

As mentioned briefly in an earlier section, we imagine that the simplistic energy models proposed in this work can eventually be used to aid the hardware development process within a specific set of efficiency goals for a set of algorithms. Thus, the intended computational kernels for a future platform can provide a basic level of hardware/software codesign at initial stages of the development process. A few more open problems to consider:

- Effect of minimizing runtime/energy given a bound on power and determining parameters given a target GFlops/W value for matrix multiplication
- Minimizing average power for the 1.5D n-body algorithm
- The effect of poor latency scaling by 2.5D LU in various processing environments (embedded, cluster, cloud)
- Accurate measurement of model parameters (especially energy-based) and comparison to measurement techniques of other researchers
- Best ways to take advantage of strong scaling regions offered by *.5D algorithms and the implications for hardware design
- Energy benefits of communication-avoiding algorithms in general (not just *.5D)
- Exploring the effect of energy constraints on embedded platforms involved in real-time critical decision making and high latency communication
- With expressions for runtime and energy, can we add a few more parameters and be able to say something about the minimum area required for a certain level of efficiency?

References

[1] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39:39–5, 1995.

[2] G. Ballard, J. Demmel, and A. Gearhart. Communication bounds for heterogeneous architectures. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*, 2011. (“brief announcement”).

[3] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. Technical Report EECS-2012-32, UC Berkeley, March 2012. To appear in SPAA 2012.

[4] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. Technical Report EECS-2012-31, UC Berkeley, March 2012. To appear in SPAA 2012.

[5] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Implementing communication-optimal parallel and sequential QR and LU factorizations, 2008.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. In *SPAA '11: Proceedings of the 23rd annual symposium on parallelism in algorithms and architectures*, pages 1–12, New York, NY, USA, 2011. ACM.

[7] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Analysis Applications*, 32(3):866–901, 2011.

[8] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M.Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of Basic Linear Algebra Subroutines (BLAS). *ACM Trans. Math. Soft.*, 28(2), June 2002.

[9] L. E. Cannon. *A cellular computer to implement the kalman filter algorithm*. PhD thesis, Bozeman, MT, USA, 1969.

[10] Intel Corporation. An introduction to the Intel QuickPath Interconnect. Online document.

[11] Intel Corporation. Intel 64 and ia-32 architectures software developer's manual. 2011.

[12] M. Driscoll, P. Koanantakool, E. Georganas, E. Solomonik, and K. Yelick. A communication optimal n-body algorithm for short- and long-range direct interactions. Technical Report In progress, EECS Department, University of California, Berkeley, 2012.

[13] R. A. Van De Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. Technical report, 1997.

[14] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, September 2004.

[15] H. Jia-Wei and H. T. Kung. I/o complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 326–333, New York, NY, USA, 1981. ACM.

[16] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008.

[17] E. Solomonik, A. Bhatele, and J. Demmel. Improving communication performance in dense linear algebra via topology aware collectives. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 77:1–77:11, New York, NY, USA, 2011. ACM.

[18] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms. Technical Report UCB/EECS-2011-72, EECS Department, University of California, Berkeley, Jun 2011.