# Perfect Strong Scaling Using No Additional Energy

James Demmel, Andrew Gearhart, Benjamin Lipshitz, and Oded Schwartz
*Electrical Engineering and Computer Sciences*
*University of California, Berkeley*
*Berkeley, USA*
$\{$*demmel,agearh,lipshitz,odedsc*$\}$@*cs.berkeley.edu*

***Abstract*—Energy efficiency of computing devices has become a dominant area of research interest in recent years. Most previous work has 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 n-body problem via the use of algorithms that use all available memory to replicate data. This means that we can increase the number of processors by some factor and decrease the runtime (both computation and communication) by the same factor, without changing the total energy use.**

***Keywords*-Energy lower bounds, Communication-avoiding algorithms, Energy efficient algorithms, Power efficiency**

## I. 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 [1] 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 exists (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 $p$) 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 as we vary the number of processors and memory size (assuming a fixed process technology). For example:

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. Given a bound on average power, can we minimize energy or runtime?
5) 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.

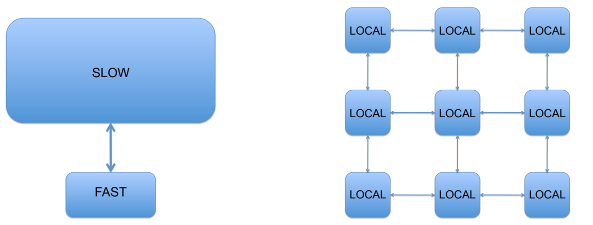
## II. Deriving Lower Bounds on Algorithm Energy Consumption

### Machine model

In this work and as in [2], we consider the same abstract model of a distributed machine shown in Figure 1(b). In this model, each processing node is homogeneous and linked within an abstract network topology. To communicate, words are packed into contiguous messages before being communicated to another processor. Synchronization is handled through messages; thus synchronization costs are part of the message count. Furthermore, as the number of processors within the distributed machine scales we assume that the link parameters (per-message and per-word costs) 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 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) Sequential model (b) Distributed parallel model

*Figure 1:* Abstract machine models

This more complicated model allows for a greater degree of parametrization by modeling two levels of machine communication (as opposed to the one-level distributed machine model primarily discussed).

## 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 $\alpha_t$ is the latency (seconds per message), $\beta_t$ 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 $\gamma_t$ 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.

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

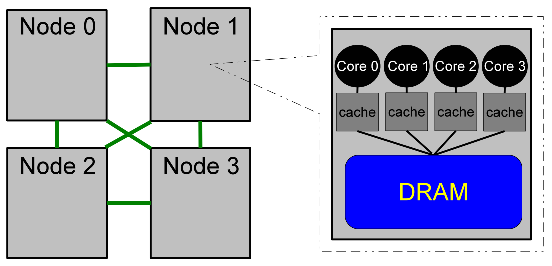


*Figure 2:* Two level machine model with 4 nodes and 4 cores per node

## 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

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

Here $\gamma_e$, $\beta_e$ and $\alpha_e$ are the energy costs (in joules) per flop, per word transferred and per message, respectively. $\delta_e$ 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, but suitable for a lower bound). $\epsilon_e$ is the energy leakage per second in the system outside the memory. Note that $\epsilon_e$ may encompass 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.

We choose to represent runtime and system energy via linear models. This is for simplicity, and reflects our goal of attempting to capture general trends in algorithm behavior to guide design efforts. Thus, extremely high model accuracy is not required to extract useful results. Interestingly, a recent paper by McCullough et al. [3] found that measuring total system power with linear models resulted in a 2-6% error for multi-core benchmarks (the paper warns about using such models for subsystem power, however). This level of accuracy is far greater than that required for this body of work. As energy consumption parameters could be influenced by more complicated processes (ex. processor heating and cooling cycles), we may consider higher-order models in future work.

## III. 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 [2], we extend the work of Hong and Kung [4] and Irony, Toledo and Tiskin [5] 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 direct linear algebra algorithms including Basic Linear Algebra Subroutine (BLAS) [6] operations (e.g. matrix-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 [2] and utilizes $M$ words of fast memory, then the total number of words $W$ sent and received by the processor satisfies

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

where $I$ and $O$ are the number of input and output words, respectively[2]. Following [2], 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 (3) by the size of the largest possible message $m$. Thus, if a processor executes $F$ flops as before, we have

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

A similar expression to Equation 3 bounds word traffic in the parallel model

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

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 dense matrix-matrix operations (LU factorization, etc.), 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, (reviewed in [2]), and have also been extended to a model of heterogeneous processing [7].

**Reducing communication by using extra memory**

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 [8] or SUMMA [9]. For reasons that will soon become clear, we refer to these algorithms as "2D". In Agarwal et al. [10], 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 [11] for more details).

In [11], 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.

A key observation regarding 2.5D matrix multiplication is that the algorithm achieves perfect strong scaling modulo $log(P)$ factors (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 to 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 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 [11] also propose an algorithm for 2.5D LU factorization that

[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.

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

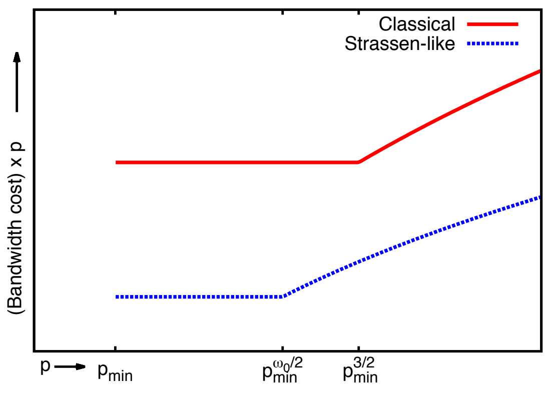


*Figure 3:* Limits of communication strong scaling for matrix multiplication. Figure adapted from [12], [13]. Note that the right portion of each line is not straight, but slightly concave.

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 proved by Ballard et al. in [12], 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 $\Theta(n^{\omega_0})$ 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.

## IV. Time and Energy Lower and Upper 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 [5], [2]:

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

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 [11], 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.

If equations (1), (2) and (8) are combined, we obtain the following lower bounds on the amount of time and energy required to run $O(n^3)$ parallel matrix multiplication, which are attained by the 2.5D algorithm[4]:

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

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

In our above discussion of 2.5D matrix multiplication, we observed that for $p_{min} = n^2/M \leq p \leq n^3/M^{\frac{3}{2}}$, 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 independent of $p$, 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

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

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 ($\beta_t$ and $\alpha_t$) and energy ($\beta_e$ and $\alpha_e$) 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 [14], 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

[4]Recall: $n^2/M \leq M \leq n^2/p^{2/3}$. Note that choosing $M, m$ equal to their maximum values $n^2/p^{2/3}$ causes a $\log p$ term to appear in the latency component. We omit this for simplicity.

the two level machine model of Figure 2 we obtain these expressions for runtime and energy

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

$$\begin{aligned} E =& n^3 \Big[ \gamma_e + \gamma_t \epsilon_e + \tfrac{\beta_e^n + \beta_t^n \epsilon_e p_\ell}{\sqrt{M_n}} + \tfrac{\beta_e^\ell + \beta_t^\ell \epsilon_e}{\sqrt{M_\ell}} \\ + & \quad \gamma_t \left( \delta_e^n \tfrac{M_n}{p_\ell} + \delta_e^\ell M_\ell \right) \\ + & \quad \left( \delta_e^n \tfrac{M_n}{p_\ell} + \delta_e^\ell M_\ell \right) \left( \tfrac{\beta_t^n p_\ell}{\sqrt{M_n}} + \tfrac{\beta_t^\ell}{\sqrt{M_\ell}} \right) \Big], \quad (12) \end{aligned}$$

where $p_n$,$\beta^n$,$\alpha^n$,$M_n$ and $\delta^n$ 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 [15], it is possible to perform fast matrix multiplication with less communication than classical matrix multiplication. The asymptotic costs are

$$F = \frac{n^{\omega_0}}{p}, \quad W = \frac{n^{\omega_0}}{pM^{\frac{\omega_0}{2}-1}}, \quad S = \frac{W}{m}.$$

These match the communication lower bounds proved in [13]. Repeating the analysis from above, we find that the total energy is:

$$\begin{aligned} E_{FLM}(n,p,M) = & \quad (\gamma_e + \gamma_t \epsilon_e) n^{\omega_0} \\ & + \Big( (\beta_e + \beta_t \epsilon_e) + \tfrac{(\alpha_e + \alpha_t \epsilon_e)}{m} \Big) \tfrac{n^{\omega_0}}{M^{\omega_0/2-1}} \\ & + \quad \delta_e \gamma_t M n^{\omega_0} \\ & + \quad \left( \delta_e \beta_t + \tfrac{\delta_e \alpha_t}{m} \right) M^{2-\omega_0/2} n_0^{\omega} \quad (13) \end{aligned}$$

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 $M = n^2/p^{2/\omega_0}$

$$\begin{aligned} E_{FUM}(n,p) = & \quad (\gamma_e + \gamma_t \epsilon_e) n^{\omega_0} \\ & + \Big( (\beta_e + \beta_t \epsilon_e) + \tfrac{(\alpha_e + \alpha_t \epsilon_e)}{m} \Big) n^2 p^{1-2/\omega_0} \\ & + \quad \delta_e \gamma_t n^5 p^{-2/\omega_0} \\ & + \quad \left( \delta_e \beta_t + \tfrac{\delta_e \alpha_t}{m} \right) n^4 p^{1-4/\omega_0} \quad (14) \end{aligned}$$

[5]With the exception of $p_l$ and $M_l$ which represent the number of cores per node and size of core local memory.

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 introduction, but analytic solutions are harder to obtain because $\omega_0$ appears in the powers of $M$.

## LU factorization

For dense LU decomposition, the 2.5D algorithm has asymptotic costs

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

It does strongly scale in the bandwidth term (which is identical, modulo constant factors, to the term in 2.5D matrix multiplication). However, it does not scale in the latency term because of the critical path. 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 associatively combine the results of individual interactions works). Like the cases of 2D and 3D linear algebra algorithms, an n-body algorithm exists that replicates data upon processors to reduce the amount of required communication [16]. The computation and communication costs for this algorithm are

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

where $n/p \le M \le n/p^{\frac{1}{2}}$ and $f$ that represents the number of flops necessary to compute the interaction of a pair of particles. Thus

$$T_{nbody}(n,p,M) = \frac{\gamma_t f n^2}{p} + \frac{\beta_t n^2}{Mp} + \frac{\alpha_t n^2}{mMp} \quad (15)$$

$$\begin{aligned} E_{nbody}(n,M) =& (f(\gamma_e + \gamma_t \epsilon_e) + \delta_e(\beta_t + \alpha_t/m))n^2 \\ + & \quad \left( (\beta_e + \beta_t \epsilon_e) + \tfrac{\alpha_e + \alpha_t \epsilon_e}{m} \right) \tfrac{n^2}{M} \\ + & \quad \delta_e \gamma_t f M n^2. \quad (16) \end{aligned}$$

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

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

$$\begin{aligned} E =& n^2 \Big[ (f\gamma_e + f\gamma_t\epsilon_e + \delta_e^n\beta_t^n + \delta_e^\ell\beta_t^\ell) \\ +& \quad (p_\ell\beta_e^n + \epsilon_e p_\ell\beta_t^n)M_n^{-1} \\ +& \ (\beta_e^\ell + \epsilon_e\beta_t^\ell)M_\ell^{-1} + \quad \tfrac{\delta_e^\ell}{p_\ell} f\gamma_t M_n \\ +& \delta_e f\gamma_t M_\ell + \tfrac{\delta_e^n\beta_t^\ell M_n}{p_\ell M_\ell} + \tfrac{\delta_e p_\ell \beta_t^n M_\ell}{M_n} \Big]. \end{aligned} \tag{17}$$

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 $\beta = \beta m + \alpha$.

**Fast Fourier transform (FFT)**

A Fast Fourier Transform (FFT) with input size $n$ performs $n \log n$ flops in $\log n$ steps. In the sequential case, a tight communication lower bound is known [4]:

$$W = \Theta\left(\frac{n \log n}{\log M}\right).$$

In the parallel case, the standard algorithm divides the data between the $p$ processors in a cyclic fashion, allowing the first $\log(n/p)$ steps to 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

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

This algorithm has recently been shown to be communication-optimal in the BSP model [17]. 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

$$\begin{aligned} E_{FFT} =& (\gamma_e + \epsilon_e\gamma_t) n \log n + (\alpha_e + \epsilon_e\alpha_t) p \log p \\ &+ (\beta_e + \epsilon_e\beta_t + \delta_e\alpha_t) n \log p + \tfrac{\delta_e\gamma_t n^2 \log n}{p} \\ &+ \qquad \tfrac{\delta_e\beta_t n^2 \log p}{p}. \end{aligned}$$

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

## V. Applications of Energy Bounds to the Direct n-body Problem

We use our model to answer several optimization questions related to energy, time, and power. In this section we show how to answer these questions for the direct n-body problem. The same techniques give qualitatively similar, but more complicated, answers in the case of classical matrix multiplication and Strassen's matrix multiplication (see details in the technical report [18]).

### A. Minimizing runtime or energy

The runtime of the algorithm decreases as $p$ or $M$ increases (see Equation 15). The minimum runtime is when $p$ is set as large as possible, and $M$ is set to be its maximum value $M = \frac{n}{\sqrt{p}}$.

For fixed $n$, total energy is minimized (see Equation 16) by using memory

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

Note that this expression is independent of $p$. Using more memory than $M_0$ is less energy efficient because of the energy cost of keeping the memory on, whereas using less memory is less energy efficient because of the increased communication cost. The minimum energy is

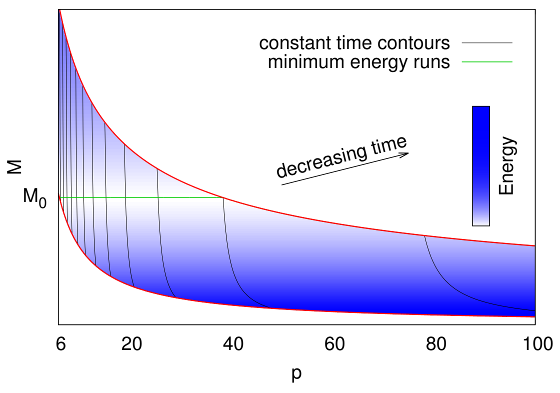
$$\begin{aligned} E^*_{nbody}(n) =& E_{nbody}(n, M_0) \\ =& n^2 \Big( f(\gamma_e + \gamma_t\epsilon_e) + \delta_e(\beta_t + \alpha_t/m) \\ &+ 2\left(\delta_e\gamma_t f\left(\beta_e + \beta_t\epsilon_e + (\alpha_e + \alpha_t\epsilon_e)/m\right)\right)^{\frac{1}{2}} \Big). \end{aligned} \tag{18}$$

It is possible to use memory $M_0$ and hence attain the minimum energy use for $p$ in the range

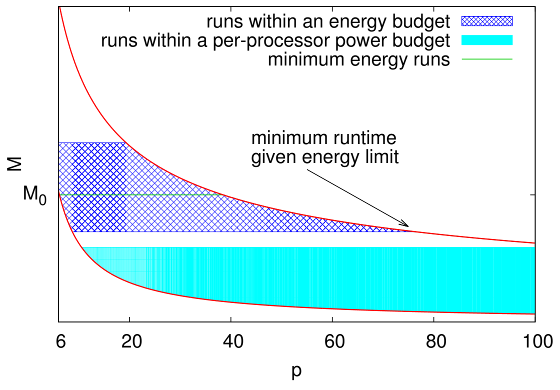
$$\frac{n}{M_0} \leq p \leq \frac{n^2}{M_0^2}.$$

In Figure 4(c) these runs are those along the green line. Using more memory than $M_0$ uses more energy because the memory uses energy, whereas using less memory than $M_0$ uses more energy because the extra communication uses more energy.

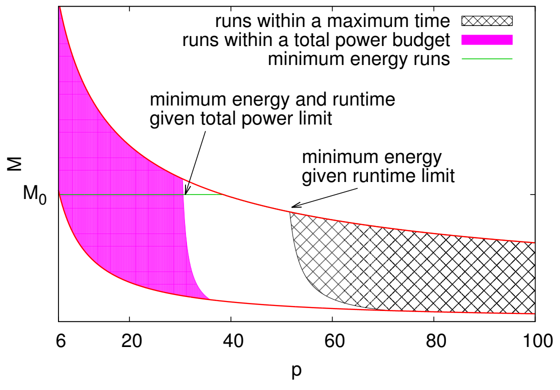
Note that minimizing energy and minimizing time select different values of $M$ and $p$. That is, in our model



(a)



(b)



(c)

*Figure 4:* Possible executions of the data-replicating n-body algorithm for a fixed $n$. The thick red lines represent the 1D and 2D limits, and the algorithm can only be run for values of $p$ and $M$ that lie in this range. The top graph (4(a)) shows the energy, which is independent of $p$, minimized at $M = M_0$, and increases if $M$ is increased or decreased from that value. It also shows lines of equally spaced constant runtime. Runtime is decreased by moving to the right or up. The middle graph (4(b)) shows which runs are possible within a given energy budget or per-processor power budget. The bottom graph (4(c)) shows which runs are possible given a maximum running time or a total power budget. The size of these regions depends on the budget. Note that these graphs are for illustrative purposes only, and use contrived parameters.

"race to halt" is not always the guiding principle for saving energy.

### B. Minimizing energy given an upper bound on the run time

An upper bound $T_{\max}$ on the run time restricts us to a region as shown in the crosshatched region in Figure 4(c). Depending on the value of $T_{\max}$, this region may or may not intersect the line of optimal energy runs. If it does, it is possible to attain the minimum possible energy $E^*_{nbody}$ within the time $T_{\max}$, otherwise the lowest energy available is at the top-left corner of the region allowed by $T_{\max}$.

Algebraically, if

$$T_{\max} \geq \gamma_t f M_0^2 + (\beta_t + (\alpha_t/m))M_0$$

then it is possible to achieve the absolute minimum energy $E^*_{nbody}(n)$ within time $T_{\max}$, for example by setting $M = M_0$, $p = n^2/M_0^2$.

Otherwise, it is necessary to use less memory than $M_0$ to be able to use enough processors to achieve the running-time bound. To be precise, it is necessary to use at least

$$p_{\min} = \left( \frac{(\beta_t + \alpha_t/m)n}{2T_{\max}} + \right.$$
$$\left. + \frac{\left((\beta_t + \alpha_t/m)^2 n^2 + 4T_{\max}\gamma_t f n^2\right)^{\frac{1}{2}}}{2T_{\max}} \right)^2$$

processors. The minimum energy to run in time at most $T_{\max}$ is attained by setting

$$p = p_{\min},$$

and running in the 2D limit $M = n/\sqrt{p}$.

### C. 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. Since $E$ depends only on $M$, not on $p$, this corresponds to a restriction to the dark blue shaded region in Figure 4(b), and the minimum time run is at the bottom-right corner of that region.

Algebraically, minimizing $T$ will always select a 2D run, since increasing $p$ from a data-replicating run until it hits the 2D boundary decreases $T$ without affecting $E$. 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 value of $p$ is given by.

$$p \leq \left( \frac{E_{\max} - An^2}{2nB} + \frac{\left( \left( -E_{\max} + An^2 \right)^2 - 4Bn^4 \delta_e \gamma_t f \right)^{\frac{1}{2}}}{2nB} \right)^2$$

where

$$A = f(\gamma_e + \gamma_t \epsilon_e) + \delta_e(\beta_t + \alpha_t/m)$$
$$B = \beta_e + \beta_t \epsilon_e + (\alpha_e + \alpha_t \epsilon_e)/m.$$

Note that this expression is has an imaginary component if the energy bound $E_{\max}$ is not attainable.

### D. Minimizing runtime or energy given a bound on total 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 $P$:

$$P_{nbody} = p \left( \frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e \right).$$

An upper bound on total power $P_{\max}^{tot}$ thus translates into an upper bound on the number of processors:

$$p \leq P_{\max}^{tot} \left( \frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e \right)^{-1}. \tag{19}$$

The fastest run will correspond to using the maximum number of processors. The most energy efficient runs correspond to using $M = M_0$ and any number of processors in the range of $n/M_0 \leq p$ and inequality (19). This case corresponds to the magenta shaded region in Figure 4(c).

### E. Minimizing runtime or energy given a bound on power per processor

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

$$M \leq \frac{C + \left( C^2 - 4\gamma_e \gamma_t f D \right)^{\frac{1}{2}}}{2\delta_e \gamma_t f} \tag{20}$$

where

$$C = \gamma_t f P_{\max} - \gamma_e f - \epsilon_e \gamma_t f - \delta_e(\beta_t + \alpha_t/m)$$
$$D = \beta_e + \alpha_e/m - (\beta_t + \alpha_t/m)P_{\max} - \epsilon_e(\beta_t + \alpha_t/m).$$

This corresponds to the cyan shaded region in Figure 4(b). To minimize $T$, we would use as many processors as possible, and as much memory as possible subject to inequality (20) and $M \leq n/\sqrt{p}$.

If $M_0$ is in the range allowed by $P_{\max}$, then the global minimum energy can be attained within a per-processor power budget $P_{\max}$. If not, since $E$ is a decreasing function of $M$ for $M < M_0$, the minimum energy is when $M$ takes its maximum value allowed by inequality 20 and $p$ is anywhere in the range $\frac{n}{M} < p < \frac{n^2}{M^2}$.

### F. 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

$$\frac{fn^2}{E^*_{nbody}},$$

where $E^*_{nbody}$ is given by equation (18). This ratio is independent of $p$, $M$, or $n$, so we get a constraint on the machine parameters $\alpha_t$, $\beta_t$, $\gamma_e$, $\alpha_e$, $\beta_e$, $\gamma_e$, $\delta_e$, $\epsilon_e$, $m$.

## VI. 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 5. This machine has two Intel Sandy Bridge server processors (code-name Jaketown) joined via Intel's Quick Path Interconnect (QPI) [19]. 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 I. To obtain $\gamma_e$, 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 $\gamma_e$, 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 $\gamma_e$, we calculate $\gamma_t$ based upon the peak single precision performance of the chip. Also, we assume 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

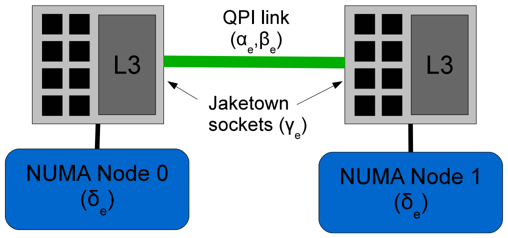


*Figure 5:* Dual-socket 8-core Intel Jaketown machine modeled within this case study.

send a message multiplied by the link power and then divided by the message length.

To gain an initial impression of the effect from scaling $\gamma_e, \beta_e$ and $\alpha_e$, 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 6 we can see that halving either $\gamma_e, \beta_e$ or $\alpha_e$ independently results in a limited amount of efficiency improvement when considering the metric of GFLOPS/W. In particular, scaling $\beta_e$ has almost no effect while the benefits of scaling $\gamma_e$ 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 multiplication and data-replicating 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 [20] 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.

*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 |
| $\gamma_e$ (J/flop) | 3.78024E-10 |
| $\beta_e$ (J/word) | 3.78024E-10 |
| $\alpha_e$ (J/msg) | 0 |
| $\delta_e$ (J/word/s) | 5.7742E-09 |
| $\epsilon_e$ (J/s) | 0 |
| $\gamma_t$ (s/flop) | 2.5202E-12 |
| $\beta_t$ (s/word) | 1.56E-10 |
| $\alpha_t$ (s/msg) | 6.00E-08 |

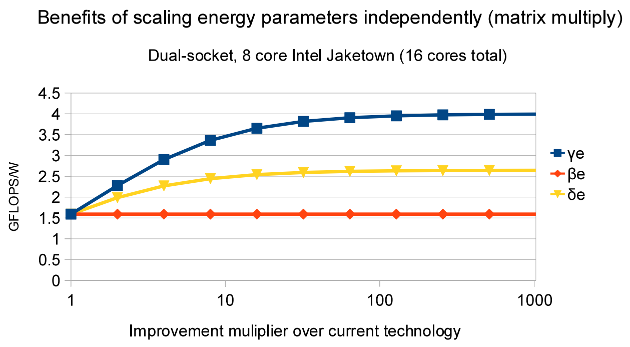


*Figure 6:* Scaling $\gamma_e$,$\beta_e$, $\delta_e$ independently on case study machine

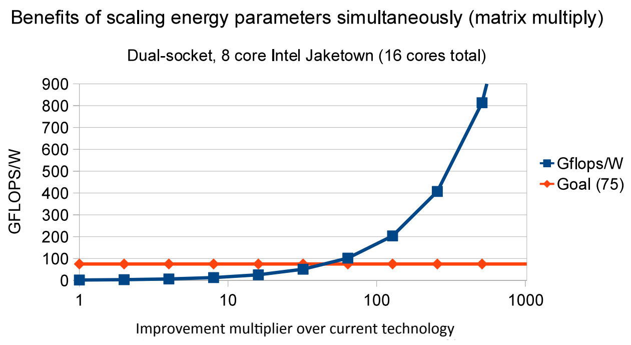


*Figure 7:* Scaling $\gamma_e$,$\beta_e$, $\delta_e$ together on case study machine

## VII. Observations and Open Problems

Table II 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, Table II highlights two poles of increased energy efficiency: high-power GPUs with high throughput and low-power slower processors. Thus, there can be a trade-off 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 hope 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 co-design at initial stages of the development process. A few more open problems include the following:

- 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 data-replicating 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
- Does the model need temperature dependent terms to accurately represent real computers?
- Best ways to take advantage of strong scaling regions offered by data-replicating algorithms and the implications for hardware design
- Energy benefits of communication-avoiding algorithms in general (not just data-replicating)
- 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?

## Acknowledgment

We acknowledge funding from Microsoft (Award #024263) and Intel (Award #024894), and matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from ParLab affiliates National Instruments, Nokia, NVIDIA, Oracle and Samsung, as well as MathWorks. Research is also supported by DOE grants DE-SC0004938, DE-SC0005136, DE-SC0003959, DE-SC0008700, and AC02-05CH11231, and DARPA grant HR0011-12-2-0016. Approved for public release; distribution is unlimited. The content of this paper does not necessarily reflect the position or the policy of the US government and no official endorsement should be inferred.

## References

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

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, “Minimizing Communication in Numerical Linear Algebra,” *SIAM J. Matrix Analysis Applications*, vol. 32, no. 3, pp. 866–901, 2011.

[3] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, “Evaluating the Effectiveness of Model-based Power Characterization,” in *USENIX ATC ’11*. Berkeley, CA, USA: USENIX Association, 2011, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002193

[4] J.-W. Hong and H. T. Kung, “I/O Complexity: The Red-Blue Pebble Game,” in *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, ser. STOC ’81. New York, NY, USA: ACM, 1981, pp. 326–333. [Online]. Available: http://doi.acm.org/10.1145/800076.802486

[5] D. Irony, S. Toledo, and A. Tiskin, “Communication Lower Bounds for Distributed-Memory Matrix Multiplication,” *J. Parallel Distrib. Comput.*, vol. 64, no. 9, pp. 1017–1026, Sep. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2004.03.021

[6] 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.*, vol. 28, no. 2, June 2002.

[7] G. Ballard, J. Demmel, and A. Gearhart, “Brief Announcement: Communication Bounds for Heterogeneous Architectures,” in *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*, 2011.

[8] L. E. Cannon, “A Cellular Computer to Implement the Kalman Filter Algorithm,” Ph.D. dissertation, Bozeman, MT, USA, 1969.

*Table II:* Example machine parameters for $\gamma_e$ and $\gamma_t$ where TDP is Thermal Design Power, FP is Floating Point, and the SIMD column represents the single-precision SIMD vector width. The Ivy Bridge processors include additional parameters for the on-package GPU.

| **Processor** | **Freq (Ghz)** | **Cores** | **SIMD** | **TDP (W)** | **Peak FP** | $\gamma_t$ **(s/flop)** | $\gamma_e$ **(J/flop)** | **Gflops/W** |
|---|---|---|---|---|---|---|---|---|
| Intel Sandy Bridge 2687W[21] | 3.1 | 8 | 8 | 150.0 | 396.80 | 2.52E-12 | 3.78E-10 | 2.645 |
| Intel Ivy Bridge 3770K[22], [23] | 3.5 (0.65) | 4(16) | 8(8) | 77.0 | 307.20 | 3.26E-12 | 2.51E-10 | 3.990 |
| Intel Ivy Bridge 3770T[24], [23] | 2.5 (0.65) | 4(16) | 8(8) | 45.0 | 243.20 | 4.11E-12 | 1.85E-10 | 5.404 |
| Intel Westmere-EX E7-8870[25] | 2.4 | 10 | 4 | 130.0 | 192.00 | 5.21E-12 | 6.77E-10 | 1.477 |
| Intel Beckton X7560[26] | 2.26 | 8 | 4 | 130.0 | 144.64 | 6.91E-12 | 8.99E-10 | 1.113 |
| Intel Atom D2500[27] | 1.86 | 2 | 4 | 10.0 | 29.76 | 3.36E-11 | 3.36E-10 | 2.976 |
| Intel Atom N2800[28] | 1.86 | 2 | 4 | 6.5 | 29.76 | 3.36E-11 | 2.18E-10 | 4.578 |
| Nvidia GTX480[29], [30] | 1.401 | 480 | 1 | 250.0 | 1344.96 | 7.44E-13 | 1.86E-10 | 5.380 |
| Nvidia GTX590[31], [30] | 1.215 | 1024 | 1 | 365.0 | 2488.32 | 4.02E-13 | 1.47E-10 | 6.817 |
| ARM Cortex A9 [32], [33] | 2 | 2 | 2 | 1.9 | 8.00 | 1.25E-10 | 2.38E-10 | 4.211 |
| ARM Cortex A9 [32], [33] | 0.8 | 2 | 2 | 0.5 | 3.20 | 3.13E-10 | 1.56E-10 | 6.400 |

[9] R. A. van de Geijn and J. Watts, "SUMMA: Scalable Universal Matrix Multiplication Algorithm," Tech. Rep., 1997.

[10] 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*, vol. 39, pp. 39–5, 1995.

[11] E. Solomonik and J. Demmel, "Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms." in *Euro-Par (2)*, 2011, pp. 90–109.

[12] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Brief Announcement: Strong Scaling of Matrix Multiplication Algorithms and Memory-Independent Communication Lower Bounds," in *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '12. New York, NY, USA: ACM, 2012, pp. 77–79. [Online]. Available: http://doi.acm.org/10.1145/2312005.2312021

[13] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Graph Expansion and Communication Costs of Fast Matrix Multiplication," *J. ACM*, vol. 59, no. 6, pp. 32:1–32:23, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2395116.2395121

[14] 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*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 77:1–77:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063487

[15] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-Optimal Parallel Algorithm for Strassen's Matrix Multiplication," in *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '12. New York, NY, USA: ACM, 2012, pp. 193–204. [Online]. Available: http://doi.acm.org/10.1145/2312005.2312044

[16] M. Driscoll, P. Koanantakool, E. Georganas, E. Solomonik, and K. Yelick, "A Communication-Optimal N-Body Algorithm for Short-Range, Direct Interactions." 2013.

[17] G. Bilardi, M. Scquizzato, and F. Silvestri, "A Lower Bound Technique for Communication on BSP with Application to the FFT," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 676–687. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32820-6_67

[18] J. Demmel, A. Gearhart, O. Schwartz, and B. Lipshitz, "Perfect Strong Scaling Using No Additional Energy," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-126, May 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-126.html

[19] Intel Corporation, "An Introduction to the Intel QuickPath Interconnect." [Online]. Available: http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[20] ——, "Intel 64 and IA-32 Architectures Software Developer's Manual," 2011. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.html

[21] Intel Corportation, "Intel® Xeon® Processor E5-2687W ," 2012. [Online]. Available: http://ark.intel.com/products/64582/Intel-Xeon-Processor-E5-2687W-20M-Cache-3_10-GHz-8_00-GTs-Intel-QPI

[22] ——, "Intel® Core™ i7-3770K Processor," 2012. [Online]. Available: http://ark.intel.com/products/65523

[23] ——, "Intel Processor Graphics Developer's Guide for 3rd Generation Intel® CoreTMProcessor Graphics on the Ivy Bridge microarchitecture," 2012. [Online]. Available: http://software.intel.com/sites/default/files/m/d/4/1/d/8/Ivy_Bridge_Graphics_Developers_Guide2.pdf

[24] ——, "Intel® Core™ i7-3770T Processor," 2012. [Online]. Available: http://ark.intel.com/products/65525

[25] ——, "Intel® Xeon® Processor E7-8870," 2010. [Online]. Available: http://ark.intel.com/products/53580/Intel-Xeon-Processor-E7-8870-(30M-Cache-2_40-GHz-6_40-GTs-Intel-QPI)

[26] ——, "Intel® Xeon® Processor X7560 ," 2010. [Online]. Available: http://ark.intel.com/products/46499/Intel-Xeon-Processor-X7560-(24M-Cache-2_26-GHz-6_40-GTs-Intel-QPI)

[27] ——, "Intel® Atom™ Processor D2500," 2011. [Online]. Available: http://ark.intel.com/products/59682/Intel-Atom-Processor-D2500-1M-Cache-1_86-GHz

[28] ——, "Intel® Atom™ Processor N2800," 2011. [Online]. Available: http://ark.intel.com/products/58917/Intel-Atom-Processor-N2800-1M-Cache-1_86-GHz

[29] NVIDIA Corporation, "GeForce GTX 480 Specifications," 2012. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications

[30] H. Hagedoom, "GeForce GTX 590 review: Product Architecture," 2011. [Online]. Available: http://www.guru3d.com/articles-pages/geforce_gtx_590_review,2.html

[31] NVIDIA Corporation, "GeForce GTX 590 Specifications," 2012. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-590/specifications

[32] ARM Holdings, "NEON," 2012. [Online]. Available: http://www.arm.com/products/processors/technologies/neon.php

[33] ——, "Cortex-A9 Processor," 2012. [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a9.php