2015 44th International Conference on Parallel Processing Workshops

Communication Avoiding Power Scaling

John Leidel
Whitacre College of Engineering
Texas Tech University
Lubbock, Texas 79409-3104
Email: john.leidel@ttu.edu

Yong Chen
Whitacre College of Engineering
Texas Tech University
Lubbock, Texas 79409-3104
Email: yong.chen@ttu.edu

***Abstract*—Recent system on chip (SoC) techniques have permitted the continued scaling of core densities at a rate sufficient to track Moore's Law. However, this continued increase in transistor density has warranted new hardware features in order to sufficiently scale the degree of on-chip concurrency. Features such as complex multi-level caches, hierarchical core configurations and hardware-assisted threading have increased the overall energy requirements of the SoC and decreased the programmer's ability to realize efficient scaling.**

This increase in overall system power requirements has resulted in research and development activities associated with hardware techniques such as dynamic frequency scaling and software techniques such as power-aware, fine-grained thread scheduling algorithms. We present the basis for a third area of research: power-scaling algorithmic complexity. The goal of this research focus is to describe techniques by which one may weigh the timing and power derivatives of competitive parallel algorithms in order to provide data necessary to make algorithmic choices based upon both the projected performance and the expected power requirements.

This work presents a model and associated technique to describe the relative energy performance scaling characteristics of parallel and mixed parallel-sequential algorithms. The model and equations are then applied to a study of matrix multiplication techniques on a symmetric multiprocessing platform. We utilize a tuned OpenBLAS blocking matrix multiplication, a classic parallel Strassen-Winograd technique and a Communication Avoiding Parallel Strassen (CAPS) technique to elicit the relative energy performance scaling on our aforementioned platform. In doing so, we show that while a blocking matrix multiplication may provide the highest potential performance on our platform, both the Strassen and CAPS techniques have ideal energy scaling properties. Furthermore, we show that by reducing the communication requirements of Strassen multiplication, we have the ability to gain a slight improvement in power scaling over traditional Strassen implementations.

I. Introduction

Given the recent advances in on-chip interconnections as well as process manufacturing techniques, system-on-chip (SoC) designs have continued to follow the logic density curves typically associated with Moore's Law. Conversely, the increase in features and architectural complexity required to maintain the scaling of logic density has negatively affected the programmer's ability to realize efficient application performance scaling [1].

The result of this phenomenon is an increase in overall SoC and system power requirements in order to realize performance in scalable applications. The increase in power requirements can be classified as those associated with on-chip arithmetic utilities, on-chip data movement (cache, et.al.), on-chip memory operations (memory controller, bus, et.al.), physical memory medium (DRAM, et.al.) and peripheral devices (DMA, PCIe, et.al.). These classifications can also be described as power domains.

This increase in overall power requirements has been identified in both traditional data center workloads as well as scientific or high performance computing workloads. Research in the areas of dynamic frequency scaling and system architecture monitoring have lead to several application and operating system-level techniques and tools to monitor and manage the overall power of the system [2] [3] [4] [5] [6] [7]. In addition to this, several attempts have been made to manage the power utilization using fine-grained thread scheduling. These techniques have lead to slight increases in overall power efficiency, but are largely based upon empirical and reactive approaches.

We present the basis for a tertiary area of research whose goal is to describe a set of techniques by which one may weigh both the performance and the energy properties of competitive parallel algorithms. The derivative results shall provide the ability to make algorithmic tradeoffs based upon the desired performance weighed alongside the total power utilization of a given parallel application on a target architecture. In doing so, we provide system architects, facilities managers and users the ability to construct and maintain scalable applications on architectures within the limits of the respective facilities while maintaining the highest potential performance or application efficiency.

This work presents a model and associated technique by which to describe the relative energy performance scaling characteristics of competitive parallel algorithmic approaches. The model and equations present both purely parallel and mixed sequential-parallel methodologies by which to characterize the power-performance scaling ratios. These equations also take into account the ability to measure and characterize individual system components, or power domains, and their affect on the energy scaling ratios. The equations provide the basis for others to develop energy performance scaling metrics for their respective architecture and algorithmic techniques.

Our equations are then applied to a series of matrix multiplication techniques on a symmetric multiprocessing system. While large-scale, dense linear algebra may not be widely applicable outside of high performance or technical computing, it provides a known basis for experimentation. We utilize a tuned OpenBLAS [8] blocking matrix multiplication routine, a class parallel Strassen-Winograd [9] technique and a Communication Avoiding Parallel Strassen (CAPS) technique [10] [11] [12]. Our conjecture in this study is that the OpenBLAS implementation may provide the highest overall performance characteristics, but does so at the detriment of power scaling. We also show that while the Strassen-derived techniques may not provide the highest performance at the relative problem sizes, their algorithmic complexity provides significantly better power scaling properties. Furthermore, we show that utilizing a communication avoiding approach such as CAPS provides the best potential for power scaling in our tests.

The remainder of this work is organized as follows. Section 2 describes the prior work in power scaling and power monitoring techniques for both traditional workloads and high performance, numerical workloads. Section 3 describes the equations by which we characterize the overall energy performance scaling of a target parallel algorithm. Section 4 describes each of the algorithmic approaches and their respective implementations. Section 5 describes our test platform and Section 6 describes our results, including the data associated with the energy performance scaling of each parallel implementation on a modern symmetric multiprocessing system. We conclude this work with further research paths in proving our approaches on systems at scale.

II. Prior Work

A. Power Measurement and Characterization

Rather than characterizing the algorithmic power performance tradeoffs, much of the recent work concerning power characteristics focuses on either dynamically measuring individual component power or dynamically controlling the frequency scaling or input voltage of individual components. Research from IBM [2] [3] and others [4] note the use of both inexpensive system-level circuitry or expensive, external monitoring components. There are also a number of recent advances in component packages associated with the Performance Application Programming Interface (PAPI) [13]. While these items provide convenient monitoring and reporting capabilities, they make no inherent effort to characterize the power utilization with respect to the target workload or algorithm.

In addition to this, there are several ongoing efforts to dynamically manage the power characteristics of a system or system component [4] [5] [6] [7]. Several methodologies and associated solutions in hardware and software have been developed in order to provide the tools necessary to manage system or component power. While these dynamic power management techniques are very useful for dynamic and manual instrumentation and modification, their core management algorithms are based upon heuristic or fundamentally reactive methodologies. This is well-suited to mixed workloads via disparate applications. However, these system-level techniques do not provide application developers the ability to make algorithmic decisions based upon the relative power performance tradeoffs.

B. Algorithmic Efficiency

Historically, high performance computing has required the use of highly efficient numerical algorithms and techniques as the basis for sequential and parallel solvers [14] [15] [16]. As such, the vast majority of classic research in numerical algorithms has focused on the raw performance characteristics therein. However, the latest generation of research into future architecture requirements [1] explicitly calls for power efficiency improvements in terms of raw hardware architecture as well as algorithm design and implementation. In this manner, these performance centric designs provide little insight into the power and theoretical power scaling tradeoffs on arbitrary computing platforms.

III. Energy Performance Scaling

In order to quantify the derivative energy effects of various algorithmic choices, we must first define the metrics by which we relate relative parallel performance to energy utilization. For any given algorithm executing over P parallel units, we define the parallel runtime as T_p. We can further define the relative energy utilization of the parallel algorithm as $EAvg_p$. Note that we explicitly avoid defining the measurement criteria and units associated with the power measurement in the high level equations. This is intended to permit flexibility in the application of the forthcoming equations. In this manner, the energy performance ratio, or EP_p, of a simple parallel algorithm can be described as:

$$EP_p = EAvg_p/T_p \tag{1}$$

We may further discretize this equation to represent the energy performance characteristics of complex algorithms that contain both sequential and parallel components. In this equation, we divide the sequential work, EP_s and its time component T_s, from the work, time and respective measured power of the P parallel units. Individual parallel units are denoted using $EAvg_p$ and T_p. From this, we have the ability to characterize the total energy performance, EP_t, as a function of the respective algorithm's sequential and parallel units.

$$EP_t = (EAvg_s + max(EAvg_p))/(T_s + max(T_p)) \tag{2}$$

The equations may be further expanded in order to permit the discrete measurement of individual and mutable architectural units. Given that different algorithms, or inner stages within an algorithm, may utilize or stress different architectural components, it is often useful to measure the performance and power characteristics of these units individually. We define the capability as the ability to identify and measure individual power planes, denoted by PPL'. Modern architectures may have the ability to measure one or more power planes. We utilize F to denote the number of potential power planes on the target architectures. All architectures shall have the ability to characterize at least one power plane. This is generally associated with the incoming system power source.

We describe the ability to encapsulate the power characteristics, $EAvg_n$, from each of the available power planes on a given platform using the following equation.

$$EAvg_n = \sum_{0}^{F} PPL'_p \tag{3}$$

Given the discretized power plane measurement equations, we may now define a fully encapsulated energy performance equation for a mixed sequential-parallel application as follows:

$$EP_t = \Bigg(\sum_{0}^{F} PPL'_s + max\Big(\sum_{0}^{F} PPL'_p\Big)\Bigg)/(T_s + max(T_p)) \tag{4}$$

Finally, we may formalize the scaling characteristics across different degrees of parallelism by applying the classic equation for scaling.

$$S = EP_p/EP_1 \tag{5}$$

Or, in expanded form:

$$S = \frac{\Big(\sum_0^F PPL'_s + max\Big(\sum_0^F PPL'_p\Big)\Big)/(T_s + max(T_p))}{\Big(\sum_0^F PPL'_s + max\Big(\sum_0^F PPL'_1\Big)\Big)/(T_s + max(T_1))} \tag{6}$$

The energy performance scaling can be graphed as a function of the empirical power and performance results. Any values that fall below the linear scaling threshold can be considered ideal in terms of power performance. This implies that as parallelism grows and performance increases, the power increases at a rate that is equal to or less than the respective performance speedup. However, if the values fall above the linear threshold, we consider the energy performance scaling to be superlinear. In this case, the performance speedup at the respective degree of parallelism requires that the system power must scale at a higher rate than the respective performance scaling. This phenomenon is depicted in Figure 1.

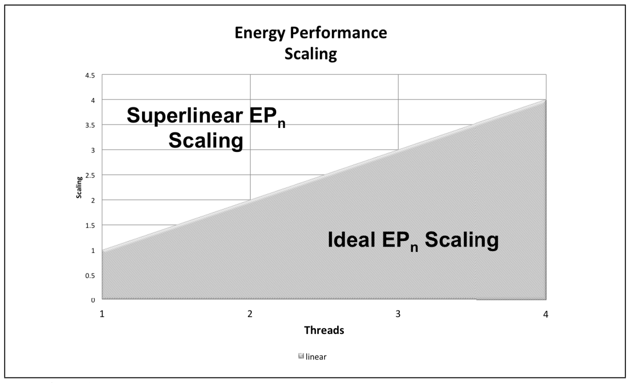

Fig. 1: Ideal and superlinear energy performance scaling

IV. Algorithms

This work utilizes three separate implementations of double precision, dense matrix multiplication. Each respective algorithm was chosen in order to exhibit a tuned, high performance implementation of the respective technique. In this manner, we optimize each of the respective test fixtures in order to mimic realistic application performance optimization.

A. OpenBLAS Tuned

The base algorithm utilized in this work is the tuned double precision matrix-matrix multiplication (DGEMM) routine as provided by the open source package OpenBLAS [8]. OpenBLAS is based upon the original work by Kazushige Goto in developing the Goto and Goto2 [17] implementations of the basic linear algebra subprograms. These implementations are highly tuned assembly routines wrapped in Fortran and C calling conventions. Each routine implements modern techniques and algorithms for modern processor and caching systems in order to make best use of processor and memory resources.

```
Data: A, B and C are NxN matrices of bxb sub-blocks;
      where b=n/N
for $i \leftarrow 1$ to $N$ do
    for $j \leftarrow 1$ to $N$ do
        Read C(i,j) into L1 cache;
        for $k \leftarrow 1$ to $N$ do
            Read A(i,k) into L1 cache;
            Read B(k,j) into L1 cache;
            C(i,j) += A(i,j) * B(k,j);
        end
        Write back C(i,j) to memory;
    end
end
```

Algorithm 1: DGEMM using blocked matrix approach

In the case of matrix-matrix multiplication, the OpenBLAS algorithm attempts to optimize a blocking matrix-matrix multiplication by determining what the best blocking factor is for the platform based upon cache hierarchy and respective capacity of each cache level. The algorithm functions especially well on square matrices of the same dimension due to its ability to coalesce like requests into the same level of the cache hierarchy. The resulting blocking DGEMM routine can be described using the pseudocode in Algorithm 1.

We compiled the OpenBLAS library on our target test platform using the target-specific compiler options and the GNU GCC Compiler tool chain version 4.8.1. At the time of this writing, we were unable to utilize generic flags for our target processor. Rather than explicitly targeting our platform's specific microarchitecture (-march=core-avx2), we utilize a configuration provided by OpenBLAS that targets the Intel Sandy Bridge microarchitecture. We also explicitly build OpenBLAS with OpenMP [18] enabled for four cores.

B. Strassen-Winograd

The second algorithm we utilize for this work is an implementation of the traditional Strassen-Winograd [9] square matrix multiplication (herein referred to as Strassen). The driving goal behind Strassen's approach and the relative derivatives is to replace the total complexity of large square matrix multiplications with a number of partitioned, smaller matrix multiplications combined with a number of matrix additions. The algorithm is applied recursively via seven intermediate operations with an accumulation to the parent sub-matrices. For each level of recursion, the partitioning schema is re-applied until the sub-matrices have converged upon a sufficiently small matrix that can be computed in the traditional manner. The end result being a reduction in overall operation count and the potential for algorithmic speedup. The basic form of the Strassen equation is given in Equation 7.

Given matrices $A_{i,j}, B_{I,j}$ and $C_{i,j}$ of dimension NxN;

$$\begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} + \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

$$\begin{aligned} Q_1 &= (A_{1,1} + A_{2,2}) * (B_{1,1} + B_{2,2}) \\ Q_2 &= (A_{2,1} + A_{2,2}) * B_{1,1} \\ Q_3 &= A_{1,1} * (B_{1,2} - B_{2,2}) \\ Q_4 &= A_{2,2} * (B_{2,1} - B_{1,1}) \\ Q_5 &= (A_{1,1} + B_{1,2}) * B_{2,2} \\ Q_6 &= (A_{2,1} - A_{1,2}) * (B_{1,1} + B_{1,2}) \\ Q_7 &= (A_{1,2} - A_{2,2}) * (B_{2,1} + B_{2,2}) \\ \\ C_{1,1} &= Q_1 + Q_4 - Q_5 + Q_7 \\ C_{1,2} &= Q_3 + Q_5 \\ C_{2,1} &= Q_2 + Q_4 \\ C_{2,2} &= Q_1 - Q_2 + Q_3 + Q_6 \end{aligned} \tag{7}$$

One of the inherent drawbacks of the Strassen approach is its inability to easily map to hardware architectures. The fundamental use of seven algorithmic stages, rather than a simple power of two, implies that Strassen shall perform best when mapped to a power of 7 processing elements. This has shown to be the case on both non-uniform shared memory systems as well as distributed memory implementations [12]. Strassen has also been known to produce differences in the numerical stability as compared with traditional techniques. A number of works have refuted the stability of Strassen as being problematic. However, these issues have been well understood [19].

Many of the higher performance Strassen implementations exploit the natural recursion in order to elicit parallelism. In the case of this work, we utilize the Strassen implementation from the Barcelona OpenMP Tasks Suite [20], or BOTS. The BOTS implementation utilizes OpenMP tasking to overlap the control flow, communication and computation portions of the Strassen algorithm, thus making it well suited as the basis for shared memory workload characterization.

The BOTS implementation of Strassen also utilizes a multi-stage recursion policy that utilizes a dense matrix solver once the recursive partitioning has generated sufficiently small sub-matrices. The dense solvers are manually unrolled such that optimal SIMD parallelism can be exploited on each child OpenMP task. After executing several empirical tests, we find that the optimal point of recursion to revert to the dense solver is when the sub-matrix Nth dimension is less than or equal to 64. As such, we utilize this cutover value across all problem sizes and thread counts.

C. Communication Avoiding Parallel Strassen

The final algorithm we utilize for this work is the Communication Avoiding Parallel Strassen, or CAPS [10] [11] [12]. CAPS is a derivative of classic Strassen methodologies as it implements a similar seven stage solver. However, the goal of the CAPS approach is to reduce both the computational complexity and the communication complexity for large square matrices.

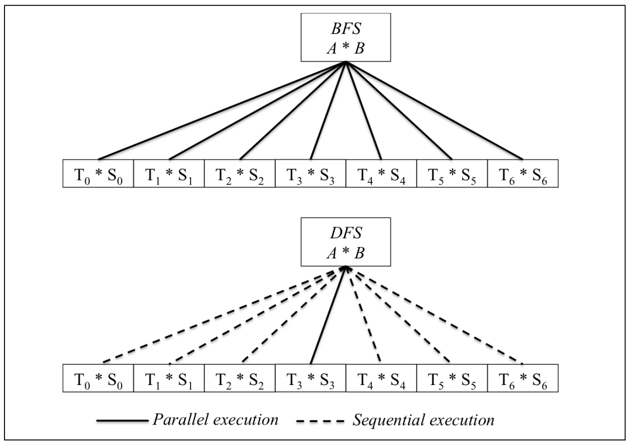

Fig. 2: Depth-first and breadth-first CAPS tree traversal

Unlike previous approaches for optimizing parallel Strassen solvers, CAPS characterizes the matrix computation in terms of traversing a tree. The tree is traversed in parallel such that at each level, determinations are made whether to recursively continue the traversal in a depth-first manner (DFS) or a breadth-first manner (BFS). The depth-first approach consists of all processing elements computing each stage of the seven Strassen sub-problems in sequence. In this manner, each stage is fully parallelized over all processing elements. The breadth-first approach requires that each of the seven sub-stages be executed in parallel on potentially disparate processing elements. The distinction between the two approaches is depicted in Figure 2. The BFS approach requires additional buffer memory in order to compute the sub-problems in a parallel manner, but it reduces the overall communication costs of the algorithm. At each level of the recursive tree traversal, the following control-flow pseudocode (Algorithm 2) is executed in order to make a determination on which algorithm to execute.

```
CAPS Control Flow;
if DEPTH ¡ CUTOFF_DEPTH then
    Execute_Strassen_BFS;
else
    Execute_Strassen_DFS;
end
```

Algorithm 2: CAPS pseudocode

While the overall algorithmic approach presented in CAPS may utilize more memory capacity, the total communication required to complete the matrix multiplication is less than classic approaches, which is summarized in Equation 8. Our conjecture in this work is that this reduction in communication not only results in higher performance, but a reduction in overall power required to execute the algorithm.

$$max\left\{\frac{n^{\omega_0}}{PM^{\omega_{0/2-1}}}, \frac{n^2}{p^{2/w_0}}\right\} \tag{8}$$

We utilize a modified version of the Barcelona OpenMP Task Suite Strassen implementation as the basis for our CAPS implementation. The breadth-first portion of the CAPS algorithm is implemented using untied OpenMP tasks. For each of the seven sub-problems, a separate task is spawned for the next level of tree recursion. The depth-first portion of the algorithm is implemented using OpenMP work sharing. Loops are parallelized such that threaded work sharing and SIMD parallelization can be realized. After much empirical testing, we utilize a cutoff depth of four to indicate the tree level at which we switch between a breadth-first and depth-first solver.

D. Algorithmic Analysis

As the basis for our algorithmic evaluation, we have chosen three algorithms that have significantly different performance characteristics based upon the scale of the input problem and the performance of the respective platform. The classic blocked (tiled) approach is known to provide near linear scaling on shared memory platforms or distributed memory platforms with low communication overhead. However, there exists a crossover point on a target platform where the Strassen-derived techniques provide better performance and the ability to exceed linear scaling due to its inherent reduction in operations. This crossover point [21] can be described for a target platform using its peak computational performance and its ability to move data (generally analogous to the memory bandwidth). We may describe the equation as follows:

Let y be the performance of a basic matrix multiplication in Mflop/s. Let z be the data movement capabilities of the target platform in MB/s. Let n be the size of one dimension of a square matrix ($n\ x\ n$) where a Strassen technique will be equivalent to the performance of competitive techniques;

$$\frac{2(n/2)^3 flop}{y\, Mflop/s} = \frac{15 * 32(n/2)^2 Bytes}{z \quad MB/s}$$

or

$$n = \frac{480 * y}{z} \tag{9}$$

Given this, the raw performance characteristics on the platform described in Section 5 are known quantities. For the purposes of this paper, we are not solely concerned with the raw performance. Rather, we focus on characterizing and measuring the relationship between performance and power for our target algorithmic methodologies.

V. Test Platform

A. Hardware

The hardware platform utilized as the basis for this work is a single Lenovo TS140 server. The original CAPS results were presented using large, distributed memory system architectures. This work focuses on power scaling of a single symmetric multiprocessor. The core processor in our system is an Intel E3-1225 (codenamed *Haswell*) quad core processor with a core frequency of 3.2Ghz and 8MB of cache. The main memory consists of a single DDR3/PC3-12800 DIMM with a bus frequency of 1600Mhz and a capacity of 4GB.

It is important to note that we disabled the default power saving features in the system BIOS. These power saving features permit the kernel and in-situ hardware logic to perform frequency scaling on cores that are not well utilized. This feature may present interesting power saving features for traditional data center applications at the detriment of a loss of accuracy when recording CPU clock ticks.

B. Software

The basis of the software configuration for this work is the OpenSUSE 13.1 Linux operating system with kernel revision 3.11.10-7 x86_64. We utilize the bundled GNU compiler tool chain (gcc 4.8.1) as delivered with OpenSUSE for all the required numerical libraries, power monitoring tools and application source code. All tests were executed as a non-privileged user. All application binaries were explicitly instrumented with the necessary permissions such that the linked libraries had the ability to access the processor MSR registers. A full list of the software revisions utilized in this work is as follows:

Software Package	**Version**	**Configuration Options**
OpenSUSE Linux	Version 13.1	Kernel 3.11.10-7 x86_64
PAPI	5.3.0	–with-components=rapl
GNU GCC	4.8.1 Build 20130909	-march=avx2 (where applicable)
Barcelona OpenMP Task Suite (BOTS)	1.1.2	CFLAGS := -march=avx2 -funroll-loops -ffast-masth -std=c99
OpenBLAS	0.2.8.0	USE_THREAD=4 TARGET= SANDYBRIDGE

TABLE I
SOFTWARE INFRASTRUCTURE

C. Power Measurement

In order to accurately measure the power and energy characteristics of our three matrix multiplication routines, we utilize the Performance Application Programming Interface [13], or PAPI. We chose PAPI over hardware-based monitoring solutions due to its inherent portability, low-perturbation to the application and flexibility to monitor other performance characteristics if desired. PAPI provides a component monitoring interface to the Intel Running Average Power Limit [22], or RAPL, library that directly reads the internal Intel processor state registers associated with power and energy utilization across the platform's power planes. The data is exported via a kernel driver to a series of event values in the Linux sys file system in */sys/bus/event_source/devices/power/events* or via an MSR values file in */dev/cpu/*/msr*.

The PAPI interface embedded in our test driver is configured to read the values from the entire package and the primary power plane (PP0) that corresponds to the CPU socket.

VI. Results

A. Execution Configuration

Our test methodology and associated execution configuration is driven by the maximum degree of hardware parallelism on our respective platform and the total capacity of main memory. Our execution matrix includes all three algorithmic approaches using randomly generated matrices of sizes {512x512, 1024x1024, 2048x2048 and 4096x4096}. Larger tests are possible using the OpenBLAS approach on our test platform. However, both Strassen-derived approaches require additional intermediate result buffers that prevent us from running problems larger than 4096x4096. Each algorithm is executed for each problem size using thread counts {1, 2, 3, 4}. Thread counts were instantiated using the *OMP_NUM_THREADS* environment variable. This provides us with 48 final result sets of algorithmic timing and performance data.

Each test was executed independently using the same driver routine with identical memory allocation schemas (GNU malloc). Tests were instantiated using a runtime script with a sleep period of 60 seconds between each test in order to quiesce the system power.

B. Performance and Scaling

As we determined in our algorithmic analysis, we fully expect both Strassen-derived approaches to perform worse than the OpenBLAS approach at the aforementioned problem sizes. The platform utilized for these tests has a relatively high compute-to-memory ratio with a relatively low memory capacity. As such, we were unable to execute problems large enough to realize the crossover point for which Strassen techniques are ideal [21].

However, we can characterize the scaling characteristics of the CAPS approach against the standard parallel Strassen implementation. The CAPS implementation performed better than the traditional Strassen test in nearly all cases across all thread counts. The communication avoiding nature of the algorithm proves to yield slightly better performance even at small problem sizes.

When compared to the OpenBLAS method, the traditional Strassen method was, on average, 2.96X slower across all problem sizes and thread counts. The CAPS implementation was, on average, 2.788X slower. This represents an average performance improvement between CAPS and our parallel Strassen of 5.97%. With this, we validate the expected performance speedup of CAPS over traditional Strassen methods. Figure 3 and Table II depict the slowdown scaling data.

Avg Slowdown	**512**	**1024**	**2048**	**4096**	**Average**
Strassen	2.872	3.477	2.874	2.637	2.965
CAPS	2.840	2.942	2.809	2.561	2.788

TABLE II
AVERAGE STRASSEN SLOWDOWN AT PROBLEM SIZE=N

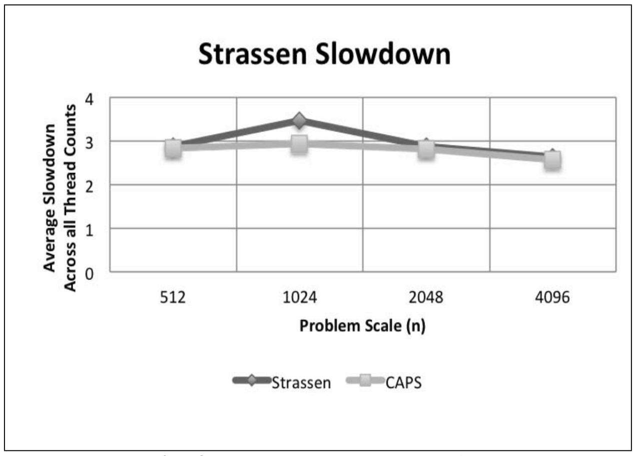

Fig. 3: Strassen slowdown scaling

C. Power Scaling

The subsequent power scaling characteristics of the three approaches was quite orthogonal to the performance characteristics. First, the OpenBLAS implementation recorded the highest power utilization on all variations of all tests. The lowest power recorded for OpenBLAS was 17.7 watts when executing the 512x512 size problem using one thread. The highest observed power for OpenBLAS was 56.4 watts when executing the 4096x4096 size problem with four threads. As shown in Figure 4, the only problem size whose power scaling was consistently near linear was the 512x512 size problem.

Conversely, the power scaling characteristics of both the Strassen (Figure 5) and the CAPS (Figure 6) implementations were sub linear across all problem sizes and all parallel thread counts. The CAPS implementation recorded a slightly lower peak power than the complementary Strassen implementation when utilizing one or two threads. The measurements were slightly higher when utilizing three or four threads. However, on average across all tests, the CAPS implementation utilized

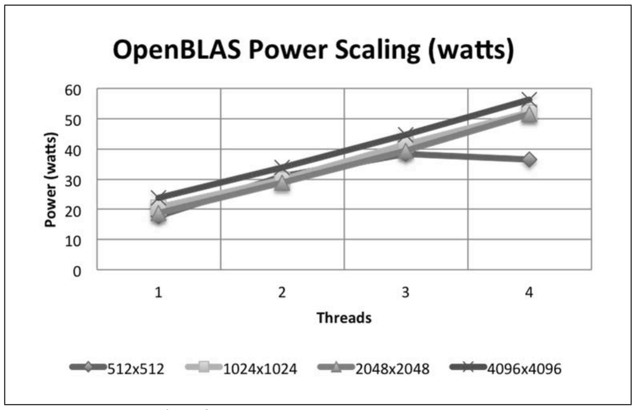

Fig. 4: OpenBLAS power scaling

26.7 watts while the Strassen implementation utilized 27.41 watts (Table III). This represents an average power improvement of 2.59%. We compare this with an average performance improvement of CAPS over Strassen of 5.97%.

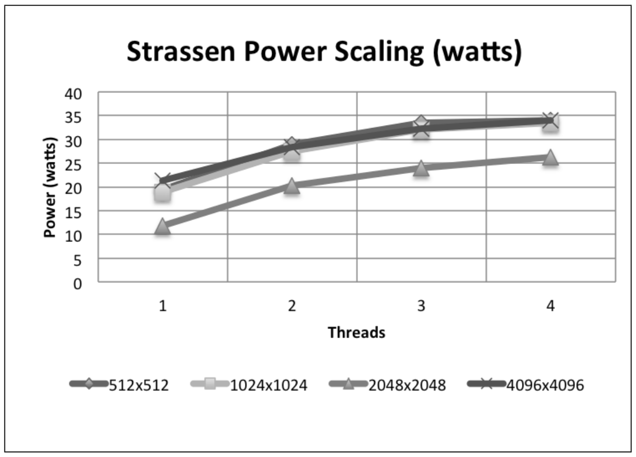

Fig. 5: Strassen power scaling

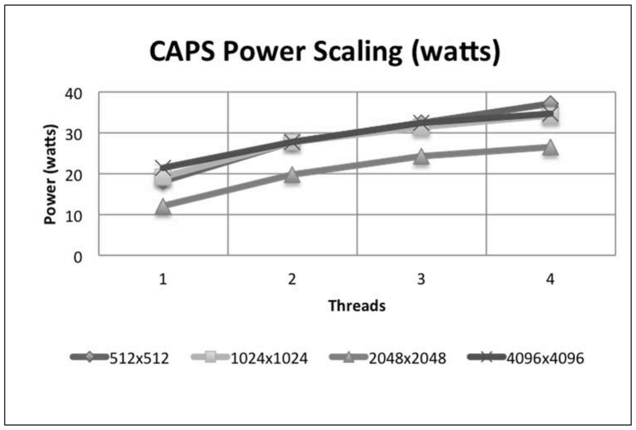

Fig. 6: CAPS power scaling

Num Threads	**1**	**2**	**3**	**4**	**Average**
OpenBLAS	20.2	30.9	40.98	49.13	35.3
Strassen	21.1	26.25	30.4	31.9	27.41
CAPS	17.7	25.75	30.175	33.175	26.7

TABLE III
AVERAGE STRASSEN SLOWDOWN AT PROBLEM SIZE=N

D. Energy Performance Evaluation

We have characterized the raw performance and raw power data from our test results, and we may apply the equations outlined in Section 3 in order to elicit the energy performance EP_n scaling of our chosen algorithms. We find that the OpenBLAS results for each problem size elicit significant scaling curves. The implication is that the power required for each additional unit of parallelism grows at a greater rate than the complementary performance advantage. The implication being, for parallel systems whose peak power is relatively limited by the local facilities, there is a significant probability the that peak parallel performance of OpenBLAS cannot be realized due to a lack of available power.

Algorithm	**512**	**1024**	**2048**	**4096**	**Average**
OpenBLAS	6356.33	1052.34	136.38	19.53	1891.15
Strassen	1912.76	239.27	24.60	4.70	545.33
CAPS	1961.28	244.57	25.32	4.86	559.00

TABLE IV
AVERAGE ENERGY PERFORMANCE AT PROBLEM SIZE=N

Conversely, the Strassen and CAPS implementations elicit relatively flat scaling curves. While their overall performance is much lower than the OpenBLAS implementation, the relative cost in terms of peak power utilization is also much lower. The implication in terms our energy performance equations is that the Strassen implementations scale performance and peak power at nearly the same rate.

Finally, we examine the total energy scaling performance across degrees of parallelism and problem sizes (Figure 7). As expected, the OpenBLAS implementation falls well beyond the linear scale in terms of energy performance scaling. As such, we can make a determination that the OpenBLAS blocked matrix multiplication implementation follows a super linear scale in terms of its energy performance ratio. Despite being the highest performance for the aforementioned problem sizes, the OpenBLAS implementation is the least optimal in terms of the energy performance ratio.

The Strassen-derived implementations elicit excellent energy performance scaling characteristics. Both the classic Strassen and the CAPS implementations have ideal or nearly ideal scaling curves. Upon further investigation, we find that our CAPS implementation is slightly closer to the linear scale than the classic Strassen implementation. The implication of this observation is that not only can we make a positive performance determination of the CAPS approach, but we may also make a positive determination regarding its energy perfor-

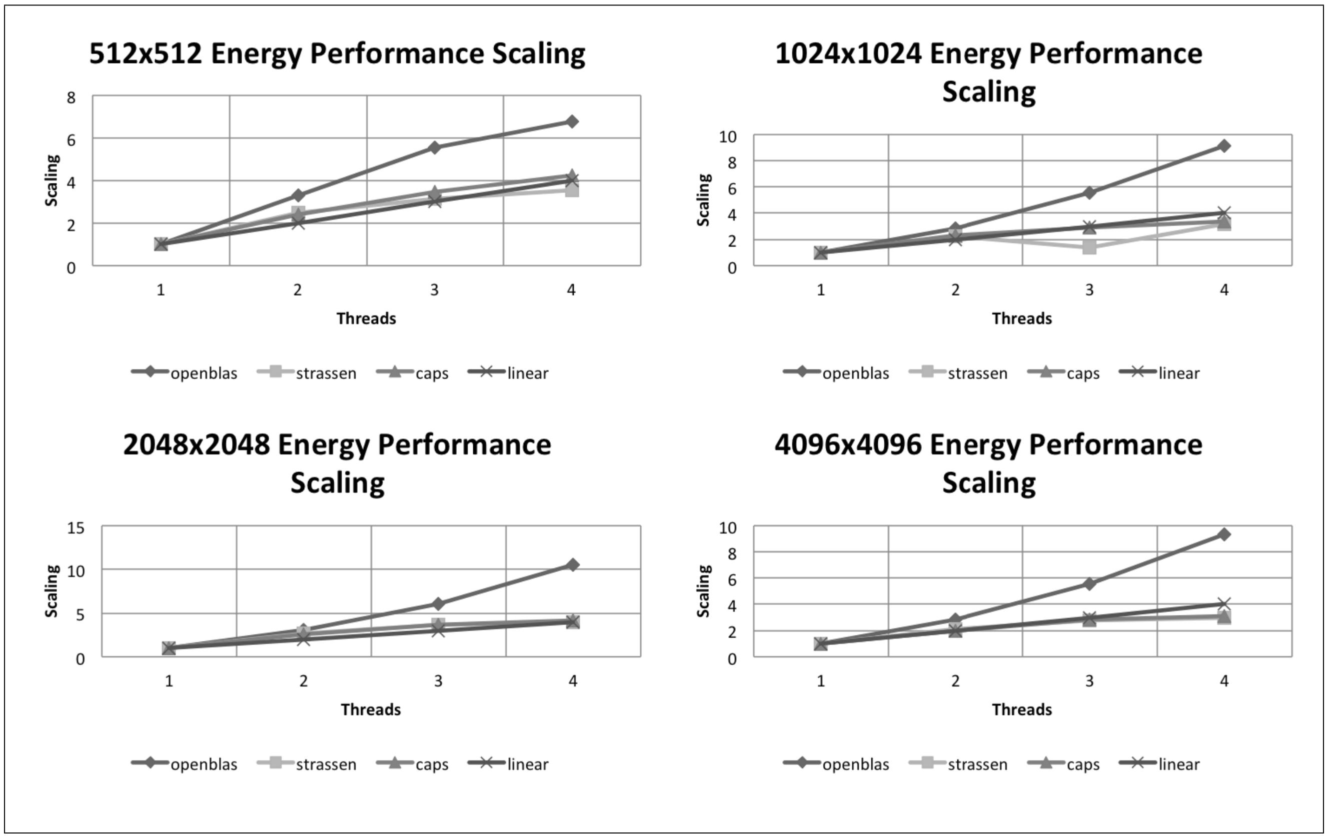

Fig. 7: Energy Performance Scaling

mance ratio. As such, we may state that the Communication Avoiding Parallel Strassen methodology is both sound in terms of higher peak performance and total energy performance utilization.

VII. Conclusions

In conclusion, we have set forth a model and a set of fundamental equations that describe the overall energy performance scaling characteristics of parallel and mixed sequential-parallel algorithms. These equations provide the ability to classify algorithmic complexities in terms of their performance and relative peak power utilization. In doing so, we provide the ability to make algorithmic determinations based upon a target problem scale, relative platform performance and peak power threshold. Given the recent focus on power efficient methodologies for monitoring and scaling system architecture component power utilization, our technique provides a tertiary path by which to maximize the power and performance efficiency of a platform a scale.

Furthermore, we utilize the aforementioned driving equations to prove the energy performance efficiency of the Communication Avoiding Parallel Strassen implementation over competitively tuned square matrix multiplication techniques. In deliberately reducing the overall communication complexity of the algorithm, the CAPS implementation provides the best energy performance scaling ratio against classic parallel Strassen methods and the blocked matrix multiplication present in OpenBLAS. As such, for sufficiently large matrices and system architectures that are sensitive to peak power utilization, we find that the CAPS implementation is most efficient.

VIII. Future Work

Given that our initial evaluation of the energy performance scaling properties of communication avoiding algorithms has proven acceptable, the next stage in our research is to continue the evaluation on larger platforms and for larger problem sizes. While our symmetric multiprocessing system has proved useful, we were unable to execute problems in excess of 4096x2096. As such, we seek to migrate the current implementation to a distributed memory implementation using MPI. Measuring the power performance characteristics of a distributed memory platform shall take into account the power associated with transmitting memory blocks across the interconnect as well as local communication traffic. In this manner, we shall build a multifaceted model of the algorithmic energy performance scaling characteristics and elicit data using larger problem sizes. We seek to utilize the same microarchitecture as utilized in this test as to make fair comparisons to the results in this work.

In addition to this, we shall also begin to quantify the energy performance scaling of a complementary set of sparse matrix multiplication techniques. However, rather than relying purely on a set of algorithmic techniques, we shall also address the energy performance scaling properties of the various sparse matrix (vector) storage techniques. In doing so, we shall provide data and results on both performance and energy scaling for a cross-section of algorithms and sparse storage techniques that can applied to future application development.

Acknowledgment

The authors would like to acknowledge the advice and guidance of Dr. Kevin Wadleigh and Dr. Isom Crawford in evaluating classic Strassen techniques.

References

[1] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead," 2008.

[2] I. Systems, "IBM PowerExecutive 1.10 Installation and Users Guide," 2006.

[3] C. Lefurgy, X. Wang, and M. Ware, "Power capping: A prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, pp. 183–195, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10586-007-0045-4

[4] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "Power aware computing," R. Graybill and R. Melhem, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, ch. The Case for Power Management in Web Servers, pp. 261–289. [Online]. Available: http://dl.acm.org/citation.cfm?id=783060.783075

[5] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. C. Rinard, "Dynamic knobs for responsive power-aware computing." in *ASPLOS*, R. Gupta and T. C. Mowry, Eds. ACM, 2011, pp. 199–212. [Online]. Available: http://dblp.uni-trier.de/db/conf/asplos/asplos2011.html#HoffmannSCMAR11

[6] Y.-H. Lu, L. Benini, and G. De Micheli, "Operating-system directed power reduction," in *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, ser. ISLPED '00. New York, NY, USA: ACM, 2000, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/344166.344189

[7] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal control techniques for power-performance management," *IEEE Micro*, vol. 25, no. 5, pp. 52–62, Sep. 2005. [Online]. Available: http://dx.doi.org/10.1109/MM.2005.87

[8] Z. Xianyi, W. Qian, and Z. Yunquan, "Model-driven level 3 blas performance optimization on loongson 3a processor," in *Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems*, ser. ICPADS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 684–691. [Online]. Available: http://dx.doi.org/10.1109/ICPADS.2012.97

[9] "Algorithms column: An overview of the recent progress on matrix multiplication by virginia vassilevska williams," *SIGACT News*, vol. 43, no. 4, pp. 57–59, Dec. 2012, reviewer-Khuller, Samir. [Online]. Available: http://doi.acm.org/10.1145/2421119.2421134

[10] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-optimal parallel aorolgorithm for strassen's matrix multiplication," in *Proceedings of the Twenty-fourth Annual 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

[11] 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

[12] B. Lipshitz, G. Ballard, J. Demmel, and O. Schwartz, "Communication-avoiding parallel strassen: Implementation and performance," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 101:1–101:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389133

[13] P. Mucci, J. Dongarra, R. Kufrin, S. Moore, F. Song, and F. Wolf, "Automating the large-scale collection and analysis of performance data on linux clusters," in *Proc. of the 5th LCI International Conference on Linux Clusters: The HPC Revolution, Austin, TX, USA*, May 2004. [Online]. Available: http://www.linuxclustersinstitute.org/conferences/archive/2004/technicalpapers.html

[14] B. Grayson, A. P. Shah, and R. A. van de Geijn, "A high performance parallel strassen implementation," Austin, TX, USA, Tech. Rep., 1995.

[15] Q. Luo and J. B. Drake, "A scalable parallel strassen's matrix multiplication algorithm for distributed-memory computers," in *Proceedings of the 1995 ACM Symposium on Applied Computing*, ser. SAC '95. New York, NY, USA: ACM, 1995, pp. 221–226. [Online]. Available: http://doi.acm.org/10.1145/315891.315965

[16] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms," in *Proceedings of the 17th International Conference on Parallel Processing - Volume Part II*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 90–109. [Online]. Available: http://dl.acm.org/citation.cfm?id=2033408.2033420

[17] K. Goto and R. Van De Geijn, "High-performance implementation of the level-3 blas," *ACM Trans. Math. Softw.*, vol. 35, no. 1, pp. 4:1–4:14, Jul. 2008. [Online]. Available: http://doi.acm.org/10.1145/1377603.1377607

[18] O. A. R. Board, "OpenMP Application Programming Interface Version 4.0.0," 2013. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

[19] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[20] A. Duran, X. Teruel, R. Ferrer, X. Martorell, and E. Ayguade, "Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp," in *Proceedings of the 2009 International Conference on Parallel Processing*, ser. ICPP '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2009.64

[21] K. Wadleigh and I. Crawford, *Software Optimization for High-performance Computing*, ser. HP Professional Series. Prentice Hall PTR, 2000. [Online]. Available: http://books.google.com/books?id=IRN0IEXJzKEC

[22] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with papi," in *Proceedings of the 2012 41st International Conference on Parallel Processing Workshops*, ser. ICPPW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 262–268. [Online]. Available: http://dx.doi.org/10.1109/ICPPW.2012.39