Novel Inexact Memory Aware Algorithm Co-design for Energy Efficient Computation

Algorithmic Principles

Guru Prakash Arumugam
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India.
guruprakash991@gmail.com

John Augustine
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India.
augustine@cse.iitm.ac.in

Eli Upfal
Department of Computer Science
Brown University
Providence, RI, USA.
eli@cs.brown.edu

Parishkrati
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India.
parishkratihhh@gmail.com

Prashanth Srikanthan
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India.
prashanthxs@gmail.com

Krishna Palem
Department of Computer Science
Rice University, Houston, USA
kvp1@rice.edu

Ayush Bhargava
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India.
ayush.bhargava15@gmail.com

Sreelatha Yenugula
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai, India.
sreelatha.yenugula@gmail.com

Abstract*—It is increasingly accepted that energy savings can be achieved by trading the accuracy of a computing system for energy gains—quite often significantly. This approach is referred to as inexact or approximate computing. Given that a significant portion of the energy in a modern general purpose processor is spent on moving data to and from storage, and that increasingly data movement contributes significantly to activity during the execution of applications, it is important to be able to develop techniques and methodologies for inexact computing in this context. To accomplish this to its fullest level, it is important to start with algorithmic specifications and alter their intrinsic design to take advantage of inexactness. This calls for a new approach to** inexact memory aware algorithm design (IMAD) or co-design. ***In this paper, we provide the theoretical foundations which include novel models as well as technical results in the form of upper and lower bounds for IMAD in the context of universally understood and canonical problems: variations of sorting, and string matching. Surprisingly, IMAD allowed us to design entirely error-free algorithms while achieving energy gain factors of 1.5 and 5 in the context of sorting and string matching when compared to their traditional (textbook) algorithms. IMAD is also amenable to theoretical analysis and we present several asymptotic bounds on energy gains.

I. Introduction

Optimizing the design of a computing system to be energy and power efficient is now accepted pretty universally as a central goal in computing system design. In fact, this pressure is felt to such a large extent that hitherto heretical approaches that allow the quality or precision of the computing system to be traded for significant energy savings are now being considered. Spanning both the hardware and software layers—please see [1] [7] and [8] for recent overviews—of a computing system stack, these approaches are increasingly found to be relevant in the context of emerging workloads such as the RMS family of benchmarks spanning recognition, mining and search [9].

On the hardware front, several artifacts of these systems such as integer and floating point units [9], FFT engines [12], neural network accelerators [15], as well instruction set level abstractions [16][18] and memories [16] have been described and evaluated. Similarly, in the software domain, machine learning based compiler and run-time abstractions [20] as well as control theoretic frameworks for managing the trade-off between quality and performance have been described [19]. In this regard, there is a lot of evidence that significant gains are possible by applying the core principles of inexact computing through empirical evidence. Other novel relationships between

978-3-9815370-4-8/DATE15/©2015 EDAA

energy and concurrency have also been described [13]. Additional background and references can also be found in our companion paper [10].

In this paper, we extend the knowledge in this emerging and exciting field by systematizing and providing a theoretical foundation through models for algorithm design that are mathematically sound and technology independent for designing and evaluating algorithms where the quality of the solution can be explicitly traded for energy gains. At this abstract level, a computing system can be thought of as being composed of a computing component and a memory, akin to the classical random access machine or RAM model [4].

Our models will emphasize the cost of accessing data from memory, as opposed to the cost of computing. In particular, we will assume that as the word size m grows, the cost of memory access grows as a function of m, whereas the cost of a computational step—addition, multiplication and other logical operations—remains a constant. Our rationale in constructing our theoretical model to be biased in this fashion to significantly emphasize the cost of accessing memory—energy or time—is to realistically reflect current technological realities. For example, we note that the energy cost of accessing memory is significantly more than that of computing, where a floating point operation consumes 1/15 of the energy of a single memory access [10]. Additionally and increasingly, emerging applications are also increasingly memory intensive—big data being the buzzword.

In this model, our memories can be rendered inexact through two different approaches. On the one hand, we can achieve memory gains by slowing down memory operations. Typically, we can achieve an energy savings that is quadratic with respect to the delay in the memory access. Alternatively, we can also achieve significant energy savings by reducing the voltage, but this comes at the cost of introducing error in our memory operations. We model this tradeoff by upper bounding the probability of error by 2^{-E}, where E is the energy provided for the memory operation.

From an engineering perspective, it will be more feasible to operate energy at either ends of the tradeoff spectra for both models. In the voltage scaled model, it is more feasible to set the delay to either the usual value or set the delay very high, which in essence, is akin to powering memory down. Similarly, in the probabilistic mode, it is more feasible to operate either at a sufficiently high voltage (where, in essence, we incur no error) or at a sufficiently low voltage (which again is akin to powering down the memory). Therefore, considering this issue of engineering feasibility, our focus in this current paper is to expose the opportunities for energy gains just by operating at these extreme ends of the tradeoff spectra. We believe that exploiting the finer tradeoffs that the models allow could lead to further gains, which we intend to explore in future.

With this as our backdrop, our main technical contribution involves using this model to demonstrate Inexactness Aware Algorithm Design (IMAD), a fundamentally new approach to algorithm and -- by extension -- application design: to expose the cost of memory accesses with inexactness as a parameter so that a new methodology of inexact memory aware algorithm design can be demonstrated. To develop and shed light on the foundations of this idea in a robust setting, we choose two classical "text book" problems, sorting and string matching [4][12].

Sorting, as we all know, plays a prominent role in almost every facet of computing and is usually one of the first algorithms encountered by any computer scientist [4]. There are two main models of computation—the comparison based model where the algorithms typically take $\Theta(n \log n)$ time, where n is the number of items to be sorted and the linear time algorithms for bounded bitwidth. String matching is also made quite prominent due to its wide application esp., in gene sequencing and analysis. Perhaps the most well-known algorithm for string matching is the Knuth-Morris-Pratt algorithm [5] that runs in $O(n + k)$ time, where n is the number of characters in the text in which we need to find a pattern of length k characters.

II. Modeling a Memory Intensive Computing System

Our goal in introducing new models comes from the fact that that the cost of memory operations dominate significantly over the cost of computation [10]. Therefore, our primary goal in designing these two models is to expose opportunities for energy gains via inexact memory aware algorithm co-design.

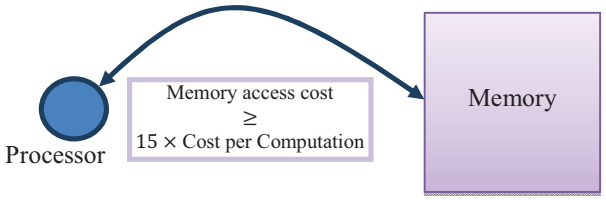

Figure 1 Dramatic cost difference between memory and computational operations.

Our two models have the same overall architecture (cf. Figure 2) with differences occurring in how we account for memory access costs. The large secondary memory contains an input array of $O(n)$ words. The width (size) of each word is m bits, m being typically (but not necessarily) larger than $\log n$. Each word is divided into the same set of non-overlapping subwords. The subwords of a word can be of different lengths, ranging from 1 (a single bit) to m (the entire word). The array of n words in the secondary memory can conceptually be thought of as columns of subwords (See Figure 2) based on the subword division of the words. Each subword column in the input array can be designed to operate under its own design parameters (including the size/width of the column, number of subword columns, delay in memory access, energy, etc.). This is the fundamental basis of the two memory models that we propose. These parameters are dynamic and can be changed at run time by the algorithm but change in parameters will incur a significant cost in energy, therefore they cannot be changed too frequently lest they become the overhead.

Building on the memory structure defined earlier, we provide two models of injecting inexactness.

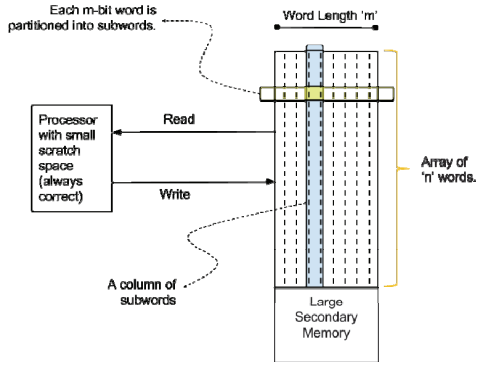

Figure 2 Common memory structure used by both models. Here, we show a single main memory bank. More generally, we could have multiple memory banks with independent inexactness parameters.

In the first model called the *voltage scaled model*, we are allowed to vary the delay in memory access for each subword column. Let the energy level required to read/write one bit with a delay one cycle be e_v. We capture the relationship between the delay c and energy consumption E is given by $E = \frac{e_v}{c^2}$. We restrict c to be bounded from above by a sufficiently large upper limit d.

In the second memory model called the *probabilistic memory model*, each subword column is associated with a read/write energy $E \geq 1$. The probability of error in each bit in the subword decreases exponentially with energy. We normalize E such that the probability of correctness $p \geq 1 - 2^{-E}$ when $1 \leq E < e_p$, for some suitably large e_p. When $E \geq e_p$, $p = 1$. Thus e_p can be viewed as the energy level at which an exact system operates.

As mentioned earlier, the algorithms we present in this paper are designed to exploit the two extreme ends of the tradeoff spectra for either models. Fortunately, these extreme ends can be unified in the following manner. We can operate each subword in two modes. In the first mode, the subword is set to operate in the exact manner and said to be ON or operating at high energy. This translates to setting the delay to one in the voltage scaled model and in the probabilistic model, this is akin to setting the energy level to e_p such that there is no error. In the second OFF mode, the subword is set to operate in a highly inexact manner and therefore, the algorithm must ignore these subwords and operate only on other subwords that are in ON mode. In the voltage scaled model, this translates to setting the delay to d (assuming d is sufficiently large), which is the maximum delay that requires the least voltage (and therefore energy). The OFF mode can be attained in the probabilistic model by setting the energy $E = 1$, which is the least possible energy level.

III. IMAD FOR SORTING

The sorting problem requires us to permute an input array of n numbers, each of word size m (assumed for simplicity to be an integral multiple of $O(\log n)$), in the secondary storage such that the permuted list is in monotonically increasing order. Alternatively, we can output a permutation of $\{1, 2, \ldots, n\}$ corresponding to the indices of the array elements in sorted order, which is the approach taken by the algorithm we present. Note that the size of the secondary storage is assumed $O(n)$ and so, the number of bits required to represent a memory address is $O(\log n)$.Thus indices and memory address pointers need $O(\log n)$ bits of storage.

A. A Variant of Radix Sort Amenable to Worst Case Analysis.

We propose a modified Radix Sort algorithm. In our version, each number in the input array is thought of as an $\left(\frac{m}{\log n}\right)$-digit number with each digit position represented by $\log n$ bits. Thus the algorithm runs in $\left(\frac{m}{\log n}\right)$ phases and in each phase the numbers are split into n buckets. Our interest is to enforce the invariant that after the i^{th} phase, the list is sorted according to the least significant $\left(\frac{i}{\log n}\right)$ bits.

To maintain n buckets, we define two arrays in secondary memory (P and Q), each of capacity n. During any given phase, exactly one of them is active and the role of active array is switched between P and Q in alternate phases (explained below). Input elements are referenced by their indices stored in the two arrays according to the current phase. Like the traditional radix sort algorithm, each bucket can be thought of as a subarray of the array. We use a third array R which stores pointers to the buckets in either P or Q, whichever is active.

In the i[th] phase, the numbers are bucketed according to the $\left((i-1)\log n\right)^{th}$ to $(i \log n)^{th}$ least significant bit positions. Let us assume without loss of generality that at the end of phase $i-1$, the indices of elements in the input array were inserted into the array P in sorted order according to first $i-1$ phases. Thus, in phase i, the array Q will become active and the indices of elements in the input array will be inserted into Q. We will now describe the steps involved in the i^{th} phase:

1. Memory phase transition: Bit positions $(i-1)\log n$ to $(i+1)\log n - 1$ are set to an energy level of e_p units. Other bits are set to an energy level of 0 units. When we read/write indices from/into arrays P, Q, or R, we will need to be careful to only use bit positions $(i-1)\log n$ to $(i+1)\log n - 1$. Note that the window of correctly read bits is $2\log n$ instead of $\log n$. This ensures an overlap of $\log n$ bits between two adjacent phases of the algorithm.
2. We now need to initialize the array R to store the starting index of each bucket in the array Q. R can be populated by traversing the input array and counting the number of elements that will get into each bucket and then computing the cumulative sum of these numbers. (Note that the indices stored in R are left shifted by $(i-1)\log n$ bits to ensure that we only use bit positions $(i-1)\log n$ to $(i+1)\log n - 1$.
3. Read the input array in the order defined by P. In each step, there are two levels of read. First an element from P

is read into a word in the scratch and right shifted by $(i-1)\log n$ bits. All bits except the last $\log n$ bits are set to 0. This element is an index which corresponds to an element in the input array. The element having this index is read from the input array into the scratch. The bits from $(i-1)\log n$ to $(i+1)\log n - 1$ gives the index of the bucket in which this element should be placed. Now we write the index of the element left shifted by $i \log n$ bits to the appropriate bucket (found from R) in the array Q. Then the corresponding element in R is incremented (by adding $2^{(i-1)(\log n)}$)

4. Switch the roles of P and Q.

At the end of this algorithm, the array P (or Q) will have sorted permutation. Since there are $O(n)$ memory operations in each phase, the total energy consumption of the algorithm is $O\left(\left(\frac{m}{\log n}\right) n e_p \log n\right) = O(nme_p)$. Time complexity of the algorithm is $O\left(nm/\log n\right)$ and the space complexity is $O(n)$.

B. Average Case Analysis of Approximate Sorting

In the variant of the radix sort algorithm described in Section III.A, if the phases *1* to p are skipped, it is easy to see that the numbers will be sorted according to $m - p\log n$ most significant bits and the energy consumption comes down by the fraction $\frac{m/logn - p}{m/logn} = 1 - \frac{plogn}{m}$. We measure the sortedness of an array A by its Kendall's τ-distance from the perfectly sorted form of A. This distance is defined as the cardinality of the set $\{(i,j)|A[i] > A[j] \wedge i < j\}$, i.e., the number of inversions in A. Thus it is an integer in the range 0 to $\binom{n}{2} \in O(n^2)$.

In the worst case, the first $m - p\log n$ bits are the same for all numbers and the input array is in reverse sorted order. For such an input the Kendall's τ-distance will be $\binom{n}{2}$. However, if the input array is drawn from a known distribution, the error may reduce dramatically. We illustrate this with the following average case analysis.

Suppose each number in the input array A is drawn uniformly at random from the range $[0, 2^m - 1]$, and we sort them according to the variant of the radix sort algorithm (in Section III.A), but ignore the first p phases. Let the indicator random variable X_{ij}= *1* if *A[i]* and *A[j]* are not in the sorted order, and X_{ij}= 0 otherwise. Thus, the Kendall's τ-distance is $\sum X_{ij}$. Since the algorithm will correctly sort numbers that differ in the first $m - p\log n$ most significant bits, for any pair i and j, $E[X_{ij}]$ cannot exceed the probability that *A[i]* and *A[j]* differ in the $m - p\log n$ most significant bits, which is $2^{-(m-plogn)}$. Therefore, we get:

Theorem 1: When each number in the input array A is drawn uniformly at random from the range $[0, 2^m - 1]$, and we sort S using the radix sort algorithm (in Section III.A), but skip the first p phases, then the Kendall's τ- distance is at most $\binom{n}{2} \times 2^{-(m-plogn)}$.

Corollary 2: If we execute only one phase in the radix sort algorithm (in Section III.A), i.e, when $p = \frac{m}{logn} - 1$, the energy consumption reduces by a factor of $\frac{logn}{m}$ and the Kendall's τ-distance reduces dramatically to $O(n)$, which means that we only have very local errors in the output.

C. Lower Bounds on Energy

Our models have a lot of similarity with I/O complexity [1][2][3] models in which lower bounds are analyzed under the indivisibility assumption in which the only set of operations allowed are transferring of an entire record between the scratch and secondary storage and comparison of two records. We adapt this definition to our context and define a notion of weak indivisibility in which no bit manipulation or compression of records is allowed in the scratch. However, a record can be divided into subwords with each subword being read from/written to the secondary storage at different energy levels/delays. We make the following claims under this assumption of weak indivisibility (but defer proofs to the full version of the paper due to space limitation).

Theorem 3: The minimum energy consumption of any correct sorting algorithm in the probabilistic model is $\Omega(nme_p)$.

Theorem 4: The minimum energy consumption of any correct sorting algorithm in the voltage scaled model with maximum delay d is $\Omega\left(nm\frac{e_v}{d^2}\right)$.

Note that these bounds are similar to the known straightforward lower bound of Ω(n) time complexity for the general sorting algorithm in the RAM model. As of today, it is unknown whether this bound is tight. However, in the energy saving architecture (both Probabilistic and Voltage Scaled models), this trivial bound on energy consumption is tight.

D. Experimental Analysis

It is interesting to note that the idea of turning off the least significant subwords that we sued in the theoretical formulation to induce inexactness have a natural interpretation in the domain of floating point numbers. To illustrate this via experimentation, we generated 100,000 floating point numbers uniformly at random from (0,1). They were stored in double-precision using 64 bits in the IEEE standard for floating point numbers (IEEE 754), in which 52 bits are used for the significand. Then, we varied the number of bits in the significand that we left ON and plotted the Kendall's τ-distance which is our error measure, using our IMAD sorting algorithms (which is our error measure). As shown in in Figure 3, IMAD shows us that we can lower the error from a very high value of 1200000 to a value close to 0 using inexact

values that are about than ten bits of the significand—as opposed to the exact representation of 52 bits. This in turn implies that the computation could have been implemented with less than half the energy in an estimated sense when compared to a computation that is exact and does not use our IMAD approach.

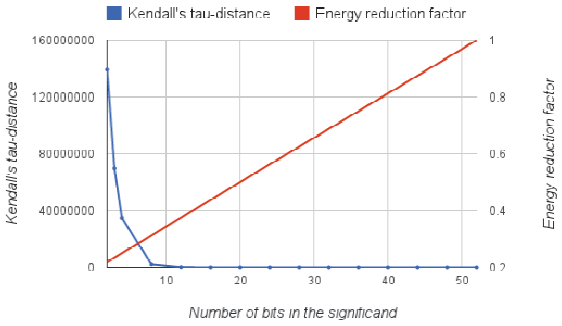

Figure 3 Plot of error and energy reduction factor as a function of the number of bits in the significand.

IV. IMAD FOR STRING MATCHING

In the string matching problem, we are given a sequence of n characters stored in contiguous memory location in the secondary storage called the *text* and a sequence of k characters stored in contiguous memory location in the secondary storage called the *pattern*. Our task is to report the index of the first occurrence of the pattern in the text. The alphabet set is assumed to be *0* to 2^m*-1*. Note that if the actual alphabet size is smaller, the characters can be stored in a more compact format wherein there could be several characters in a single word. For simplicity, we restrict to the case of one character per word. In later sections, we explain briefly how to extend these ideas to the multiple characters per word case.

A. General Technique

The key idea in designing string matching algorithms for this architecture is to observe that for a pattern to match a piece of text, a subset of bits of the pattern must match with the corresponding set of bits of the text. Since the architecture gives us the ability to read/write select subwords, we can eliminate several index positions by checking only a subword in each word. Once we have a small set of *candidate* indexes, the entire word can be checked to find an exact match.

Formally, we define a parameter $p \leq m$. In the first pass of the algorithm, the $m - p$ MSBs are ignored. This corresponds to a string matching algorithm run on a smaller character set. If there is a match, we mark the index as a potential match (candidate). Alternatively, to avoid adversarial inputs (last p bits matching but not the rest), a random set of p bits can be used instead of the LSBs.

We describe our ideas using one standard string matching algorithm, namely, the Knuth-Morris-Pratt (KMP) [5] algorithm. To illustrate how these ideas can be used in other string matching algorithms, we also briefly discuss a variation using the Rabin-Karp algorithm.

B. KMP Algorithm

The algorithm can be divided into two phases. In the first phase, a set of index positions are marked as candidates by looking at only a subword. In the second phase, each of these indices are checked for an exact match by looking at the entire word. To elaborate a bit, in the first phase, a subset of p bits is chosen and energy level is set to e_p units. The rest of the bits are set at an energy level of 0 units. An array is initialized in the secondary storage which stores the starting index of the candidate matches. The standard KMP algorithm is run on the given text and pattern but the operation of checking equality of two characters is modified to the following: If the subset of p bits in the two characters are equal, then the characters are considered equal. Any match reported by this KMP algorithm will be added to the candidate array. Note that each element of the candidate array and the internal array used in KMP algorithm stores an index position. Since an index requires $\log n$ bits of storage, p is restricted to be $\log n$. However, by storing a single index across many words, this restriction can be relaxed.

In the second phase, since we need to check for an exact match, the energy level is set to $e_p$units of energy for all bits. We first run the algorithm to generate the *partial match* table (also known as *failure* function) of the pattern. In the search phase of the algorithm, since we have already eliminated some index locations, sometimes the algorithm can 'skip' a portion of the text.

Note that if the pattern is expected to be found very early in the text, the above approach of generating candidates throughout the text is inefficient. In such cases, the algorithm can be modified to switch back and forth between the two phases by splitting the text into many small parts. Alternatively, the switch can be made once the number of entries in candidate array exceeds a threshold.

C. Analysis

The energy consumption in the first phase of the algorithm is clearly $O(p(n+k)e_p) = O(pne_p)$ since $n \geq k$. For the second phase, the energy consumption depends on the Candidate array. In the worst case, there won't be any skip operations and so the energy consumption will be $O(nme_p)$. Since $p \leq m$, the total energy consumption in the worst case is $O(nme_p)$which matches the energy consumption of running KMP directly. However, in practice, the number of elements in the candidate array will be significant lower than n and the energy consumption will come down. Note that when multiple characters fit a word, the energy consumption will be $O(nbe_p)$ where b is the number of bits per character.

D. Experimentation

As noted above, although the worst case analysis suggests that the energy consumption is not improved, in practice, the energy consumption is much lower. To illustrate this, we experimented with the book "Harry Potter and the Order of Phoenix" as the source for various patterns. The energy consumption of running KMP directly is compared with the KMP approach designed using IMAD. The searches are made

for patterns not found in the text or found towards the end of the text. Since we are working with English alphabet, the number of bits per character is 8. The results are plotted in Figure 4.

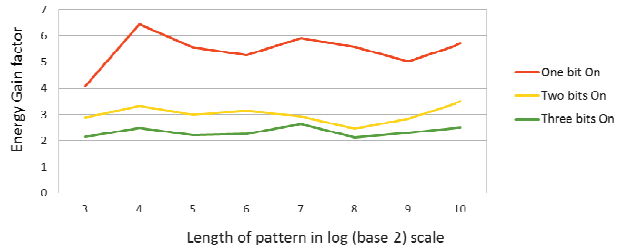

Figure 4 Energy gain factors are shown for the inexact implementation of the KMP algorithm for a variety of pattern lengths. The experiments were performed with 1, 2, and 3 bits turned on. Note that despite the use of inexact computing techniques, there is no error in string matching.

E. Rabin-Karp Algorithm

As another example, the ideas described above can be extended to Rabin-Karp algorithm [12] as well. The algorithm can be divided into two phases. In the first phase, the hash is computed only on the integer value corresponding to the chosen subset of p bits. This will give a set of candidate matching positions in the text. In the second phase, Rabin-Karp algorithm can be run on all the bits but when two adjacent elements from candidate array are far apart, instead of computing the hash value in a rolling fashion, the hash can be recomputed for the next element directly.

F. Multiple characters per word

If several characters can fit into one word, we can no longer choose any random subset of bits in each word to high energy level since the algorithm requires that the subset of bits positions checked for each character to be the same. For example, if $m = 64$ bits and alphabet size is 8 bits, and if the bit positions 0 and 1 are set to an energy level of e_p, it is required that the bit positions 8, 9, 16, 17, 24, 25… to be set to energy level of e_p as well. Additionally, since the candidate array stores an index to a character in the text, the number of bits set at high energy level must be $\log\left(\frac{n+m}{r}\right)$ where r is the number of bits per character.

V. Conclusion

We have introduced inexact memory aware algorithm design and presented several relevant algorithms. We believe that we have only scratched the surface and a lot of important improvements to our models and algorithms lie ahead of us. A more detailed technical report (with all the proofs intact) will be made available shortly. We believe that a good synergy between architecture and algorithm designers is crucial to exploiting the full benefits that inexactness has to offer.

VI. References

[1] A. Aggarwal, B. Alpern, A. K. Chandra, M. Snir, "A Model for Hierarchical Memory," STOC 1987: 305-314.

[2] A. Aggarwal, A. K. Chandra, M. Snir, "Hierarchical Memory with Block Transfer," FOCS 1987: 204-216.

[3] A. Aggarwal and J. S. Vitter. 1988, "The input/output complexity of sorting and related problems," *Commun. ACM* 31, 9 (Sept. 1988), 1116-1127.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms," MIT Press, 2009.

[5] D. Knuth, J. Morris, Jr., and V. Pratt, "Fast pattern matching in strings," SIAM Journal on Computing, 6(2):323–350, 1977.

[6] K. Palem, and A. Lingamneni, "Ten years of building broken chips: the physics and engineering of inexact computing," ACM Transactions on Embedded Computing Systems (TECS), 12(2s), Article 87, 2013.

[7] K. Palem, "Inexactness and a future of computing," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 372(2018), 2014.

[8] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," In 18th IEEE European Test Symposium, 2013, pp. 1-6.

[9] Y. K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, ..., and M. Smelyanskiy, "Convergence of recognition, mining, and synthesis workloads and its implications," In Proceedings of the IEEE, 96(5), 2008, 790-807.

[10] P. Duben, J. Schlachter, Parishkrati, S. Yenugula, J. Augustine, C. Enz, K. Palem, T. N. Palerm, "Opportunities for Energy Efficient Computing: A Study of Inexact General Purpose Processors for High-Performance and Big-data Applications," to appear in DATE 2015.

[11] A. Lingamneni, C. Enz, J. L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," In DATE, 2011, pp. 1-6.

[12] R. M. Karp, M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development* , vol.31, no.2, March 1987, pp. 249--260.

[13] K. Palem, "The arrow of time through the lens of computing," In *Time for verification*, Zohar Manna and Doron A. Peled (Eds.). LNCS, Vol. 6200. Springer-Verlag, Berlin, Heidelberg, 2010, pp. 362-369.

[14] K. Palem and A. Lingamneni, "What to do about the end of Moore's law, probably!" In DAC, 2012, pp. 924-929.

[15] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," ASP-DAC '14, 201-206.

[16] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. "Approximate storage in solid-state memories," In MICRO, 2014, pp. 25-36.

[17] S. Venkataramani, S. T. Chakradhar, K. Roy, A. Raghunathan, "Approximate computing for efficient information processing. In Embedded Systems for Real-time Multimedia (ESTIMedia)," 12th Symposium on IEEE, 2014, pp. 9-10.

[18] S. Venkataramani, A. Ranjan, K. Roy, A. Raghunathan, "AxNN: energy-efficient neuromorphic systems using approximate computing," In ISLPED 2014, pp. 27-32.

[19] X. Sui, A. Lenharth, D. Fussell, K. Pingali. Tuning variable-fidelity programs. Submitted for publication.

[20] Hoffmann, H. CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems. In ECRTS 2014.