Spatio-Temporal Locality Characterization

Andreea Anghel *, Gero Dittmann *, Rik Jongerius † and Ronald P. Luijten *
* IBM Research – Zurich, Switzerland
† IBM Research, Netherlands

***Abstract*—Reducing data communication over memory is one of the main current engineering challenges. To achieve this, optimization done solely at the application level is no longer sufficient. Holistic design-space exploration is necessary to evaluate the suitability of hardware architectures for a given application. Furthermore, a software-hardware co-design is needed to be able to leverage the inherent locality of programs and reduce data motion in the system. For this to be possible, the locality has to first be properly quantified. In this work, we propose a new approach to characterizing both the spatial and the temporal locality of a program using a hardware-architecture agnostic, generalized reuse distance metric. We propose a locality characterization metric in the form of a probability distribution as well as a visualization tool that is able to express this metric in an easily understandable way. We apply our new approach to the study of several algorithms. We show that the output of our visualization tool in the form of 2D locality plots, that we refer to as *locality heat-maps* or simply *heat-maps*, is able to clearly express the program's spatial locality, even in cases where existing metrics have proven less insightful. By quantifying both the spatial and temporal aspects of locality, the approach we propose can thus supply useful input not only to cache memory design, but also to systems based on emerging technologies such as non-volatile memory, systems that aim at bringing computation closer to data.**

I. Introduction

We are currently experiencing a shift of the performance bottlenecks in a system in that more and more applications are becoming increasingly less bounded by their computation side and increasingly more bounded by the speed at which data can be retrieved/written from/to memory. This is a direct consequence of the increasingly larger difference between processor and memory speeds. The necessity of a data-centric system design approach instead of a traditional compute-centric one is apparent if we look at the predicted gap between I/O and compute power consumption of future computing systems. Shalf et al. [1] anticipate that, by the end of this decade, reading an operand from DRAM will consume at least 100 times more energy than a floating-point operation. Efficient data motion will soon represent an unavoidable engineering challenge and, therefore, it will become essential for applications to increase their awareness of the underlying memory subsystem and using that awareness, enhance the way they move data to exploit locality patterns as much as possible.

Many memory system models and program optimizations have been proposed to address these issues. Most of them have used cache hit rate as a metric to optimize the usage of the cache memory system. Indeed, cache memory systems work efficiently if subsequent memory accesses exhibit good locality of references. Locality has two dimensions, a spatial and a temporal one, and both impact application performance. While *spatial locality* refers to the tendency of applications to reference memory addresses that are near other recently accessed addresses [2], *temporal locality* quantifies the amount of reuse of the same memory location during application execution. In order to analyze the impact of a program's locality on performance, we first need to quantify locality and, if possible, efficiently visualize it. In this paper we propose a generalization of the stack reuse distance metric proposed in [2] to jointly characterize the program's spatial and temporal locality patterns. While stack reuse distance is mainly useful to reason on the cache architecture that would best fit a certain memory access pattern (e.g. cache line size), our generalized metric can further be used and leveraged by memory prefetchers which can bring data from the memory closer to the CPUs ahead of time.

II. Related Work

The data locality properties of programs have been extensively studied and modeled in the context of both memory design and code optimization. Beyond qualitative descriptions provided in computer architecture books [3], various locality characterization metrics have been proposed in literature. In terms of temporal locality, Pyo et al. [4] introduce *reference distance* as the total number of references between accesses to the same data. Beyls et al. [5] show that this metric cannot exactly predict cache behavior for fully associative caches, but an alternative metric is able to do so. This metric is called *reuse distance* or *stack reuse distance* [2] and is defined as the total number of distinct data elements accessed between two consecutive references to the same element. The same study [5] uses this locality characterization metric to analyze the distribution of the conflict and capacity misses in the execution of code generated by an EPIC compiler, and reasons on the impact of increasing parallelism in an application on the number of capacity misses.

In terms of spatial locality, previous works have attempted to quantify it via mainly scalar metrics that allow for easy ordering and/or clustering of applications in locality classes. This has been done for example by using some form of reduction function to aggregate distances between consecutive or close to consecutive memory accesses [6] or by looking at locality from the perspective of the efficiency of cache line usage [7]. All these studies however tend to treat the spatial and temporal dimensions of locality as completely orthogonal to each other, and thus only offer a pair of uni-dimensional,

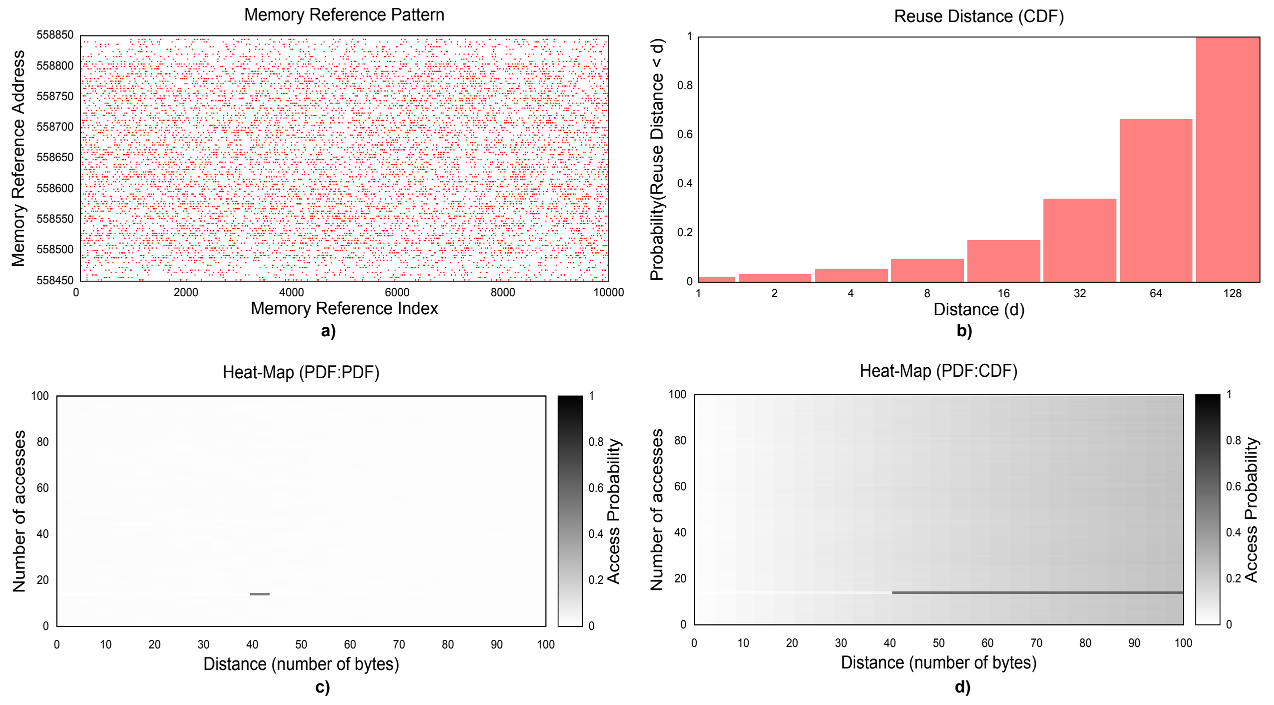

Fig. 1. Three locality representations for the *customized random access* algorithm: a) *Scatter plot* – it shows a set of consecutive memory addresses accessed over time; b) *Cumulative reuse distance distribution plot* – the reuse distance is defined as in [2]; c) *Spatio-temporal locality heat-map (PDF:PDF)*; d) *Spatio-temporal locality heat-map (PDF:CDF)*. For the latter two representations, the Y-axis represents the temporal dimension of the newly proposed metric (by the number of consecutive accesses to memory), while the X-axis represents the spatial component of the metric (in number of bytes). The intensity of a given heat-map "pixel" is proportional to the value of the $p(s,t)$ metric defined in Sec. III.

often even scalar views on the way applications handle data accesses. We propose to generalize these concepts and quantify accurately the entire two-dimensional spatio-temporal locality characteristic of a program. While this approach has the drawback of replacing a small set of values with a *locality signature* for each application, thus making it difficult to reason on application categories, it also offers a much more complete view of the application properties, allowing for increased optimization potential, e.g., either via memory system/cache design or prefetching strategies.

III. Method Description

Our workload model considers an application processing a given input as an ordered list of instructions, each instruction possibly incurring a memory reference. We filter the list for instructions entailing a memory reference. In the resulting ordered list, we assign subsequent instructions consecutive indexes, starting with index 0. Thus the index is a measure of temporality, as instructions with consecutive indexes will be executed consecutively in time (without any other memory affecting instructions in between). Hence, every entry in the list is thus characterized by at least the following three parameters:

- i - memory reference index;
- a - memory reference start address;
- n - memory reference length (in bytes).

The list serves as an input to the analysis. We quantify the spatio-temporal locality as follows. To cover both the spatial and temporal aspect, we seek a measure of the likelihood that, given that a memory location is accessed at some point during the execution of the application, a memory location situated at a fixed distance s (in number of bytes) from the original is accessed at a fixed distance in time t (in terms of the number of memory-accessing instructions). As individual memory references typically affect several bytes, we generalize this goal to this case by considering the likelihood of a reference that is s bytes away from *any* of the individual original byte locations. Formally, given a memory reference (i,a,n) at some point during the execution of an application, we seek to measure $p(s,t)$, defined as the probability that the next t-th memory reference of the application $(i+t, a'=a+s, n')$ will have the property detailed in Eq. 1.

$$(a \leq a' \text{ and } a+n > a') \text{ or } (a > a' \text{ and } a < a'+n') \quad (1)$$

In this case $p(s,t)$ is defined for $t \geq 0$ and $s \geq 0$ and is always a real number between 0 and 1. We compute this probability set as follows. For each value of t, we extract every pair of accesses that are at a distance t from each other in the memory reference list, and then for every possible value of s, we count how many of these pairs satisfy the property given by Eq. 1. Let that number be $k_{s,t}$. Then $p(s,t) = k_{s,t}/(|L|-t)$, where L is the list of references and $|L|$ is its length.

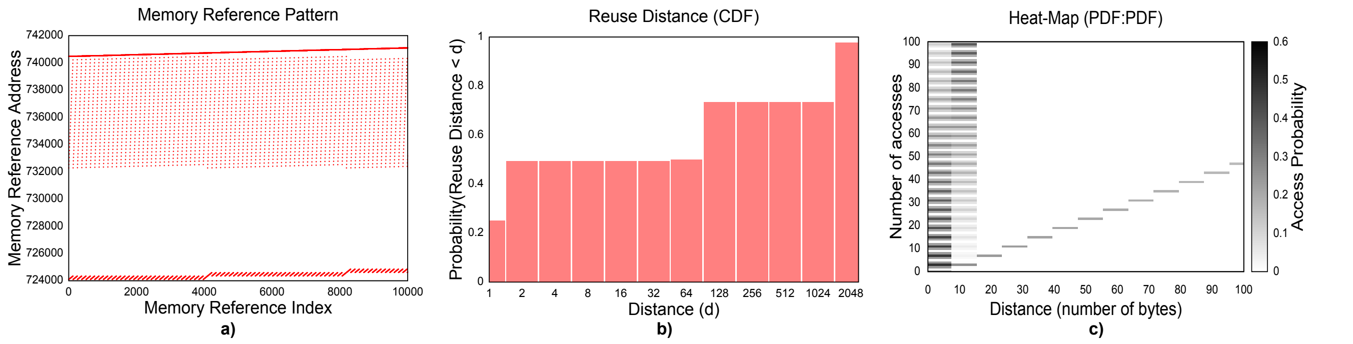

Fig. 2. Three locality representations for the *matrix multiplication* algorithm: a) *Scatter plot*; b) *Cumulative reuse distance distribution plot*; c) *Spatio-temporal locality heat-map (PDF:PDF)*.

IV. Implementation Details

We use the LLVM compiler infrastructure [8] to analyze the program's load/store memory activity. LLVM emulates a machine with an unlimited number of registers and allows us to measure application properties based on true data read-after-write data dependencies. Moreover, it permits us to reason on application performance/properties across different hardware platforms with only one single LLVM execution.

We extended the LLVM interpreter with a memory analyzer module which monitors the memory references during a C/C++ program's execution. Thus, we extract the full trace of memory references together with their corresponding access indexes and memory reference bit lengths, as required by the model explained in Sec. III. We filter from the memory reference list the references that on a real system would translate into actual memory system (RAM, cache) read/write operations. For example, accesses to temporary local variables (e.g. for loop indexes) would typically be performed via processor registers and not implicate the memory systems at all.

The LLVM memory trace is input to a separate C++ analysis application that computes the locality probability distributions ($p(s,t)$) and creates the 2D locality heat-map that shows the program's both spatial and temporal locality in one single graph. More precisely, the heat-map shows the probability of accessing a memory address at a distance of exactly $Distance$ bytes from the currently accessed memory address in the next exactly N memory references. We call this representation a PDF-PDF (probability distribution function) spatio-temporal locality – Fig. 1c). Variations of this plot are possible, e.g. the PDF-CDF (cumulative distribution function) plot shows the probability of accessing a memory address at a distance of exactly $Distance$ bytes from the currently accessed memory address in one of the next N memory references – Fig. 1d) – or the CDF-PDF which shows the probability of accessing a memory address at a distance of at least $Distance$ bytes from the currently accessed memory address in the next exactly N memory references.

V. Proof of Concept

To show the benefits of our 2D locality heat-maps and visualization method, we first implemented a proof of concept algorithm that clearly exhibits spatial locality. We call this algorithm *customized random access*. We designed the algorithm as shown in the code sample below, where v is a vector of $N = 100$ elements. For every $t + 1 = 15 + 1 = 16$ memory accesses, the same vector element address is referenced.

```
m = 3000; N = 100; d = 10; t = 15;

for(int j = 0; j < m; j++){
 int increment = 0, sum = 0;

 for(int k = 0; k < t; k++){
  int random = ((double) rand())/(RAND_MAX+1E-6)
  var(k) = random*(N-d);
  sum = sum + v[var(k)];
 }

 for(int k = 0; k < t; k++)
  sum = sum + v[var(k)+d+increment++];}
```

Fig. 1 shows how the locality of this program is represented using two commonly used approaches and our locality heat-maps. The first scatter representation – Fig. 1a) – fails to capture any locality pattern. On the other hand, the second plot in Fig. 1b), the reuse distance distributions, exposes some of the temporal locality characteristics, but fails to accurately quantify spatial locality. Finally, our proposed spatio-temporal representations – Fig. 1c) and d) – precisely indicate that there is a significant probability of accessing a memory location which is $4 \cdot 10 = 40$ bytes away from a current access, in the next 16 memory accesses and that the probabilities of other locality patterns are significantly lower.

Given a certain cache line size L, the reuse distance corresponding to it captures only an aggregate measure of the first L columns of our representation, and is thus severely limited in its expressivity of locality, especially in terms of spatial locality and spatio-temporal locality correlations.

VI. Preliminary Results

We apply our locality analysis methodology to a well-known algorithm, i.e. matrix multiplication. We use one matrix multiplication kernel of size 32. Fig. 2 shows the same three

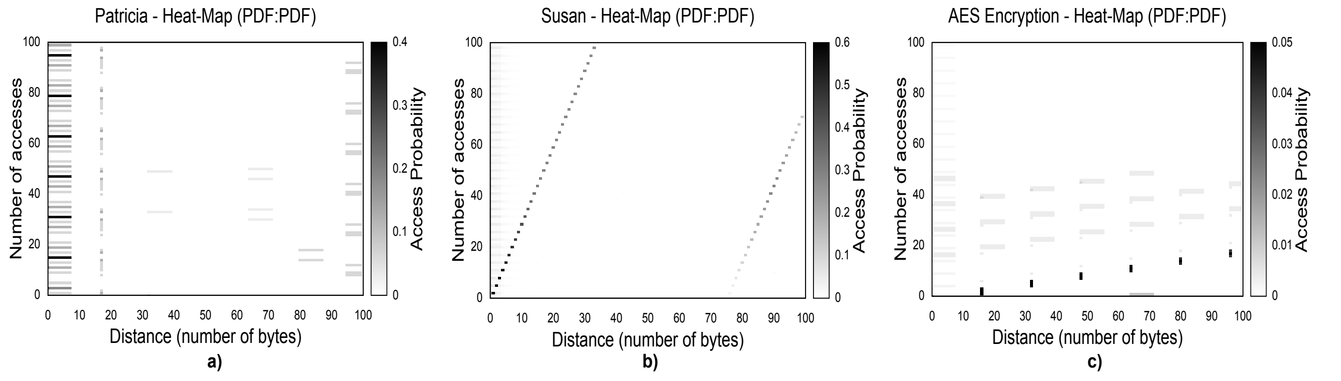

Fig. 3. Spatio-temporal locality heat-maps (PDF:PDF) for three MiBench [9] benchmark algorithms. a) Patricia (networking algorithm); b). Susan (image processing algorithm); c). AES (encryption algorithm).

representations of locality for this algorithm as Fig. 1. Our heat-map is able to accurately capture the increasing levels of spatial locality that characterize this application whereas the other two metrics fail to do so. Indeed, we can notice two distinct locality patterns. One corresponds to the $0-15$ spatial range where both the reuse distance metric and the 2D locality heat-map approach are able to identify memory reuse patterns that occur with a probability between approx. 0.15 and 0.4. Even for this first pattern, our approach provides significantly more granularity. But there is also a second pattern that consists of there being a reasonably high probability (in the $0.1-0.2$ range) for an access to happen at a memory reference $2X$ bytes further from a current access (and with a stride of 8 bytes), in the $X-$th subsequent access. The reuse distance metric fails to capture this pattern and, consequently, the prefetching optimization that it enables, whereas our approach makes it evident.

The method can be easily applied to most algorithms. In Fig. 3 we show a few examples of the kinds of visualizations that can be obtained. We applied the method to 3 algorithms from the MiBench benchmark suite:

- Patricia is an algorithm used in network applications for IP prefix lookup. Its corresponding locality heat-map is shown in Fig. 3a).
- Susan is an image recognition package for corners and edges detection. We analyze the smoothing capability of this algorithm for locality patterns shown in Fig. 3b).
- Rijndael encryption (Advanced Encryption Standard) is a block cipher with the option of 128-, 192-, and 256-bit keys and blocks. We show the locality heat-map for 128-bit key and block in Fig. 3c).

For each of them, we notice that there are locality patterns to be observed in the region for which a simple reuse distance based analysis does not provide visibility into. In general, this new method will bring most benefit compared to current state-of-the-art for applications that exhibit a strong spatial locality characteristic.

VII. Practical Applicability

The original goal of our locality analysis was to enable holistic system design-space exploration. The locality distributions that are captured during run-time analysis were meant to be used in a subsequent offline processing step to generate coarse-grained bounds for full-system performance, power consumption and cost estimation. This is achieved by using a set of mathematical formulas that capture the interaction between the properties inherent to the application and the hardware characteristics of a system –cache line size, cache size, cache associativity etc. In this context, application properties need not refer only to spatial and temporal locality (in the form of the distributions presented herein), but can also include parallelism at different granularities, i.e., instruction-level, thread-level or data-level or other aspects.

The method however is much more generic and has potentially multi-fold applicability for system design. When needing to reach a certain design goal, it is often the case that there is a choice to be made between either different applications to achieve that goal, different implementations of the same application or even different compiler or runtime optimizations for a given implementation. Our visualization technique allows us to compare these different ways to achieve the goal from a memory-reuse-potential perspective and identify the best application/implementation/optimization.

Another application of the method lies in the domain of prefetching. Our tool can show for example what type of memory access pattern an application has and thus help in determining what type of prefetching would enhance performance most. Badawy et al. [10] presents an in-depth study of various prefetching techniques and their relative performance on several popular algorithms. If the pattern is relatively regular (e.g. with affine array accesses) hardware-prefetching techniques could be useful to address the CPU-memory performance gap. Otherwise, if the pattern is iregular, software-prefetching could be a better solution to hide memory latencies, provided that the system has enough memory bandwidth.

Furthermore, the probability distribution generated by our tool can be interfaced more intimately with a prefetching engine and serve as the basis for that engine's decisions of when and what to bring from memory ahead of time.

Finally, although until now we have mainly focused on single-system analysis, where locality is used to bridge the gap between fast caches and slow RAM memory, the tool can also be used in the context of distributed systems, where applications run on several nodes and locality can be used to bridge the gap between fast local processing and slow remote data operations. The memory access distribution can be used to answer such questions as for example what the optimum amount of local memory is that would ensure a certain ratio of local to remote processing.

For example, [11] presents an FPGA-based system that moves computation close to data to greatly reduce the amount of data that must travel across a slow I/O interface. The system offloads data and I/O intensive application code to SSD to exploit the low latency and high internal bandwidth of non-volatile memory (NVM) technology. Using the temporal CDF - spatial PDF version of the locality heat-map of an application offloaded to the SSD, and assuming an interval T of accesses for which we wish to guarantee a certain ratio between computation accessing local versus remote data, we can use the row corresponding to T in the heat-map to derive the optimal SSD block size. Indeed, if the distribution is quasi null past some value S on the spatial axis, then a block size equal to S would already guarantee that a kernel running on one storage processor would not need to access, within that time frame, data farther away than in one of its two neighbors as well as roughly a $1:1$ ratio of local to remote data accesses.

Finally, in the context of the same FPGA-based system, and similarly to identifying data that can be prefetched to the processor, our locality information could be used to identify data that needs to be processed in the next accesses and initiate the appropriate processing thread on the memory device or NVM controller. Such research directions will be further explored in future work.

VIII. Conclusions

We have presented a locality characterization method for quantifying and visualizing locality patterns in applications and have shown several examples of how the approach applies to real algorithms. We have shown that our 2D locality heat-maps capture and provide useful information about both the spatial and temporal locality of a program, thus an improvement over the reuse distance metric. The locality probability distributions provided by our method can be used by memory prefetchers to hide memory access latencies caused by transfers across the memory hierarchy and can serve as a core enabler for holistic software-hardware co-design, including in the field of emerging data-centric technologies.

As future work, we plan a complete evaluation in terms of locality of various benchmark suites such as SPEC, NAS Parallel and Graph 500 and reason on the locality similarity of different algorithms. We will also relate the measure of locality obtained with the LLVM compiler infrastructure to actual hardware program executions. Finally, we will analyze the interaction between program parallelism and locality in order to understand what the best trade-off is between parallelism and the cost of data movement.

IX. Acknowledgements

This work was conducted in the context of the joint ASTRON and IBM DOME project [12] and is funded by the Netherlands Organisation for Scientific Research (NWO), the Dutch ministry of EL&I, and by the province of Drenthe.

References

[1] J. Shalf, S. Kamil, L. Oliker, and D. Skinner, “Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect,” in *SC ’05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 17.

[2] C. Ding and Y. Zhong, “Predicting whole-program locality through reuse distance analysis,” *SIGPLAN Not.*, vol. 38, no. 5, pp. 245–257, May 2003.

[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

[4] P. Changwoo, L. Kyung-Woo, H. Hye-Kyung, and L. Gyungho, “Reference distance as a metric for data locality,” in *Proceedings of HPC-ASIA97*, 1997, pp. 151–156.

[5] K. Beyls and E. H. DHollander, “Reuse distance as a metric for cache behavior,” in *Proceedings of the IASTED conference on Parallel and Distributed Computing and Systems*, 2001, pp. 617–662.

[6] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, “Quantifying locality in the memory access patterns of HPC applications,” in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, ser. SC ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 50–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.59

[7] R. C. Murphy and P. M. Kogge, “On the memory access patterns of supercomputer applications: Benchmark selection and its implications,” *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 937–945, 2007.

[8] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” in *Proceedings of the international symposium on Code Generation and Optimization: feedback-directed and runtime optimization*, ser. CGO ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 75–88.

[9] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” *Workload Characterization, Annual IEEE International Workshop*, vol. 0, pp. 3–14, 2001.

[10] A.-H. A. Badawy, A. Aggarwal, D. Yeung, and C.-W. Tseng, “The efficacy of software prefetching and locality optimizations on future memory systems.” *J. Instruction-Level Parallelism*, vol. 6, 2004. [Online]. Available: http://dblp.uni-trier.de/db/journals/jilp/jilp6.html#BadawyAYT04

[11] A. De, M. Gokhale, R. Gupta, and S. Swanson, “Minerva: Accelerating data analysis in next-generation SSDs,” *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 0, pp. 9–16, 2013.

[12] P. C. Broekema, A.-J. Boonstra, V. C. Cabezas, T. Engbersen, H. Holties, J. Jelitto, R. P. Luijten, P. Maat, R. V. van Nieuwpoort, R. Nijboer, J. W. Romein, and B. J. Offrein, “DOME: Towards the ASTRON and IBM Center for Exascale Technology,” in *Proceedings of the 2012 Workshop on High-Performance Computing for Astronomy Date*. New York, NY, USA: ACM, 2012.