Memory Profiling for Intra-Application Data-Communication Quantification: A Survey

Imran Ashraf, Mottaqiallah Taouil, Koen Bertels
Computer Engineering Lab, TU Delft, The Netherlands
Email: {I.Ashraf, M.Taouil, K.L.M.Bertels}@TUDelft.nl

***Abstract*—With the advent of technology, multi-core architectures are prevalent in embedded, general-purpose as well as high-performance computing. Efficient utilization of these platforms in an architecture agnostic way is an extremely challenging task. Hence, profiling tools are essential for programmers to optimize the applications for these architectures and understand the bottlenecks. Typical bottlenecks are irregular memory-access patterns and data-communication among cores which may reduce anticipated performance improvement. In this study, we first survey the memory-access optimization profilers. Thereafter, we provide a detailed comparison of data-communication profilers and highlight their strong and weak aspects. Finally, recommendations for improving existing data-communication profilers and/or designing future ones are thoroughly discussed.**

I. Introduction

Although the number of transistors per chip is growing due to the technology scaling [1], increasing the clock rate of processors is becoming economically less viable due to fabrication cost and power consumption [2]. These limitations shifted the trend towards the integration of a growing number of homogeneous or heterogeneous processing cores in general-purpose, embedded and high-performance computing platforms. However, these multi-core architectures pose specific challenges regarding their programmability, as the effective utilization of these platforms in an architecture agnostic way is not possible. Hardware constraints, such as memory bandwidth, local scratch-pad memory etc. need to be explicitly taken into account.

There is a huge code base of legacy, sequential applications which need to be ported to such multi-core platforms, and thus need to be parallelized. To port an existing sequential application to multi-core platform, applications must be divided into smaller parts which are mapped to the available cores in the architecture. This is a critical task, as an improper partitioning and mapping may result in performance degradation. Main identifiable reasons are irregular memory-access patterns and the communication among cores which may reduce the anticipated performance improvement.

Because of the growing gap between processor and memory speed [3], it is becoming increasingly important to optimize memory-access behavior of an application. Secondly, with the growing number of cores, the degradation of performance improvement exacerbates, as communication is typically more time-consuming than computation. Hence, this communication is considered as the major design challenge in multi-core architectures[4] and a major source of energy consumption[5].

Another important concern is the growing complexity of programs as the demand of processing is increasing in various computing domains. Hence, program analysis tools are required to highlight the hot spots and/or bottlenecks of the programs pertaining to the architecture at hand [4]. Apart from intra-function memory-access patterns, these tools should also provide the inter-function data-communication information.

Profilers are program analysis tools which provide information about various aspects of programs. For instance, number and types of instructions, frequency of function calls, time consumed per function call, etc. A lot of work is available in literature for profilers that focus at the fine granularity of instructions or at the coarse granularity of individual functions [6], [7], [8]. Cache profiling, which is a kind of memory profiling [9], has also been studied extensively. However, very few tools exist which provide intra-application data-communication information. Therefore, in this work our focus is on memory profiling tools with a focus on those which provide intra-application data-communication information. Even though there is no generally acknowledged classification of memory profilers, we propose to organize the discussion based on the following three aspects of profiling:

- Profiling objective (Section II)
- Profiling input (Section III)
- Profiling technique (Section IV)

The remainder of this paper is organized as follows. We start by detailing the proposed classification in Section II, Section III and Section IV. In Section V, we compare existing memory profilers which provide the data-communication information. To the best of our knowledge, we have included all the tools in this regard. Based on this study, we provide some recommendations for the improvements of the existing data-communication profilers or design of the future ones, in Section VI. Finally, Section VII concludes the paper.

II. Memory Profilers based on Profiling Objective

Based on the profiling objective, we further classify memory profilers into four classes, namely, memory-access optimization, memory debugging, dependence analysis and workload characterization, depicted in Figure 1. Furthermore, memory-access optimization profilers are further classified

978-1-4673-9994-4/15/$31.00 ©2015 IEEE

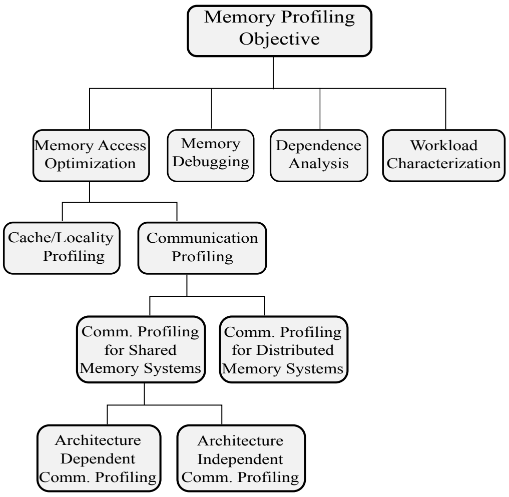
Memory Profiling Objective
Memory Access Optimization
Memory Debugging
Dependence Analysis
Workload Characterization
Cache/Locality Profiling
Communication Profiling
Comm. Profiling for Shared Memory Systems
Comm. Profiling for Distributed Memory Systems
Architecture Dependent Comm. Profiling
Architecture Independent Comm. Profiling

Fig. 1. Classification of memory profilers based on profiling objective.

into cache/locality profilers and data-communication profiling. The focus of this work is on profilers designed for memory-access optimization. We believe that the other classes of profilers have been discussed in other studies and a number of example tools are available.

A. *Memory profiling for memory-access optimization*

Profilers in this class analyze performance issues related to memory accesses in applications. For instance, the performance of an application may suffer because of pure locality of memory accesses. Cache/locality profilers can highlight the parts of the application responsible for such behavior. Another aspect of performance optimization is the communication among the parts of applications running on separate homogeneous/heterogeneous cores. Data-communication profilers provide such information and highlight communication related performance bottlenecks.

1) Locality/cache profilers: In order to decrease the gap between processor and main memory, small but very fast memories, known as caches, are used. As the size of these memories is small, only the most frequently used data can be stored in these memories based on the prediction of the algorithm in cache controller. Hence, analysis of cache behavior is crucial to increase the performance of the programs, and/or design new cache algorithms. Various tools exist which profile applications to analyze cache behavior as listed in Table I. A few well-known open-source tools are detailed below.

Cachegrind [10] is Valgrind tool which can detect first and last level instruction and data cache misses for C/C++ programs. Cachegrind tracks cache statistics (I1, D1 and L2 hits and misses) for every individual line of source code executed by the program. At program termination, it prints a summary of global statistics, and dumps the line-by-line information to a file. This information can then be used by an accompanying script to annotate the original source code with per-line cache statistics. KCachegrind, a visualization tool for the profiling data generated by Cachegrind, is also available.

Oprofile [11] is a hardware dependent, open-source profiling tool that works on recording events from hardware performance measurement units. Apart from various other performance events, it can sample events related to $L1$, $L2$ instruction and data caches to provide information about cache hits/misses by an application on a certain platform.

NumaTOP [12] is an open-source tool for runtime memory locality characterization and analysis of processes and threads running on a NUMA system. It utilizes Intel hardware performance counters sampling technologies to identify where NUMA related performance bottlenecks reside. This performance data is associated with Linux runtime information to provide real-time analysis in a GUI on production systems.

2) Data-communication profilers: Memory profilers in this class, profile applications to measure communication among various parts of an application. These profilers are further classified in to the following two classes.

a) Shared Memory Data-communication Profilers: Table II provides a summary of such profilers. QUAD (Quantitative Usage Analysis of Data) [21] provides dynamic information regarding data usage between any pair of co-operating functions in an application. This tool is based on Pin [25] and it tracks the reads and writes to a memory location at the granularity of byte. When a function writes to a memory location, it is saved as a producer of this memory location in a Trie data structure. The function reading this memory location is called the consumer and by getting the information from the Trie, a producer-consumer communication relationship is established. Apart from providing the quantitative information about the number of bytes, two other metrics are also reported. The first metric is the number of unique memory addresses, while the second metric is the number of data values uniquely communicated from producer to consumer.

PINCOMM [22], [26] is a tool based on Pin [25] which constructs Dynamic Data Flow Graph (DDFG) to report the communication flow between various parts of the program. The parts can be functions, data structures, threads etc. which are represented on DDFG. The communication is reported in the form of producer-consumer relationship. The information can also be provided in terms of marked region in the code which appear on the DDFG. These markers can also be used to start and stop communication. The dynamic objects allocated during the execution of the program are also detected to report the communication through these objects.

CETA (Communication Extraction from Threaded Applications) [23] provides data-flow information between multiple threads. Memory reads and writes are tracked at runtime using Simics multiprocessor architecture simulator [27]. Hash table is utilized to record the writing thread of an address. When the read is performed the communication is updated in another hash table. After the completion of simulation, Python scripts report the collected information as a DOT graph.

Redux [24] is a Valgrind based tool for drawing the detailed dynamic data flow graphs of programs. Because of these details, it can only be used for small kernels or parts of programs, as discussed by authors. Secondly, the purpose of the tool as

TABLE I
CACHE/LOCALITY PROFILERS

Profiler	Input			Output	CL/IDE	Technique	Based on	Availability	Supported Platform	
	Src/Binary	Language	ST/MT						OS	Architecture
Cachegrind[10]	Binary	NA	MT	Text Reports, Graphical Reports	CL, IDE	DBI	Valgrind	Open-Src	Linux, OSX Android	Intel, AMD, PPC
Oprofile[11]	Binary	NA	MT	Text Reports	CL	HWC	Arch. Perf. Counters	Open-Src	Linux	Intel, AMD, ARM, PPC, IBM
NUMATop[12]	Binary	NA	MT	Text Reports	CL	HWC	Arch. Perf. Counters (Intel PMU)	Open-Src	Linux	Intel
Dprof[13]	Binary	NA	MT	Text Reports	CL	HWC	Arch. Perf. Counters (AMD IBS)	Open-Src	Linux	AMD
Zoom[14]	Binary	NA	MT	Graphical Reports	CL, IDE	HWC	Arch. Perf. Counters	Free	Linux, OSX Windows	Intel, AMD, ARM
Vtune[15]	Src for detailed graphical reports	C, C++, C#, Java Fortran, OpenMP, MPI, OpenCL	MT	Graphical Reports	CL, IDE	HWC, DBI	Arch. Perf. Counters (Intel PMU), Pin	Commercial (Intel)	Linux, Win	Intel
Graphical Program Analysis Toolkit[16]	Src for detailed graphical reports	C, C++, pthreads	MT	Graphical Reports	CL, IDE	SBI	ATOM	Commercial (HP)	Tru64	Intel

Caliper[17]	Src for detailed graphical reports	C, C++	MT	Graphical Reports	CL, IDE	DBI	-	Commercial(HP)
								Free(Non-Commercial)
									HP-UX	HP Integrity Servers

CodeXL [18](successor (of Code Analyst)	Src for detailed graphical reports	C, C++, OpenCL	MT	Graphical Reports	CL, IDE	HWC	Arch. Perf. Counters (AMD IBS, TBS)	Commercial(AMD), Open-Src (Linux)	Linux, Win	AMD
Visual Profiler[19]	Src for detailed graphical reports	C, C++, C#, C++ AMP, Visual Basic, Visual F#, Java Script, OpenMP	MT	Graphical Reports	CL, IDE	HWC	Arch. Perf. Counters	Free	Win	Intel, AMD
Solaris Studio[20]	Src for detailed graphical reports	C, C++, OpenMP, MPI, Java	MT	Graphical Reports	IDE	HWC	Arch. Perf. Counters, VampirTrace(MPI)	Free	Solaris, RHEL	SPARC, X86-64

Src: Source, **ST:** Single Threaded, **MT:** Multi threaded, **DBI**: Dynamic Binary Instrumentation, **SBI**: Static Binary Instrumentation, **CL**: Command line, **IDE**: Integreted Development Environment, **HWC**: Hardware Counters, **IBS**: Instruction Based Sampling, **TBS**: Time Based Sampling, **PMU**: Performance Monitoring Unit

TABLE II
DATA-COMMUNICATION PROFILERS

Profiler	Input			Output	CL/IDE	Technique	Based on	Availability	Supported Platform	
	Src/binary	Language	ST/MT						OS	Architecture
QUAD[21]	binary	NA	ST	DOT, XML	CL	DBI	Intel Pin	Open-Src	Win, Linux, Android, OS X	Intel, ARM
Pincomm[22]	binary	NA	MT	CSV	CL	DBI	Intel Pin	Open-Src	Win, Linux, Android, OS X	Intel, ARM
CETA[23]	binary	NA	ST	DOT	CL	architecture simulation	Virtutech Simics	Open-Src	Win, Linux	Intel
Redux[24]	binary	NA	ST	text	CL	DBI	Valgrind	Open-Src	Linux, Android, OS X	Intel, AMD, ARM, PPC

Src: Source, **ST:** Single Threaded, **MT:** Multi threaded, **DBI**: Dynamic Binary Instrumentation, **CL**: Command line, **IDE**: Integreted Development Environment

reported by authors is to represent the computational history of a program and not the communication behavior.

b) Distributed Memory Data-communication Profilers: Message Passing Interface (MPI) [28] is a popular example of distributed memory programming model. The MPI provides communication functionality between a set of processes in a language independent way. This explicit communication is carried out through routines like `MPI_send` and `MPI_recieve`. Various commercial [29], [20], [30] and well maintained open-source [31], [32], [33], [34] tools exist which track these routines to characterize communication in MPI programs. We refer the reader to the comparative studies [35], [36] for further details. We would like to highlight here that these tools are not designed to provide the communication profile of sequential applications. These tools are based on the technique which requires MPI parallel program as input. Hence, it only helps in validating the parallel program written only in MPI, rather than constructing one.

III. MEMORY PROFILERS BASED ON INPUT APPLICATION

Memory profilers can take sequential or parallel application as an input to provide the memory access behavior of an application. Profilers also exist which can profile both sequential and parallel applications.

A. *Profiling sequential applications*

Profilers in this class profile sequential applications for performance analysis, report communication, trace bugs, detect data-races etc. QUAD [21] is an example of such profilers.

B. *Profiling parallel applications*

Profilers in this class provide information about the memory access behavior of the parallel applications. ParallelTracer [37] is a trace-based performance analysis framework for heterogeneous multicore systems. It instruments source code to trace various events in the application. It is an extension of Trace Collection and Trace Post Processing (TCPP) framework [38]. Furthermore, pin based tools [39], such as Parallel Amplifier, are available for the analysis and optimization of parallel C/C++ programs.

IV. MEMORY PROFILERS BASED ON THE PROFILING TECHNIQUE

Based on the technique, memory profilers are broadly classified as static and dynamic analysis tools as shown in Figure 2. *Static analysis* tools provide the information based on the source-code without running the application. These tools can predict the communication in regularly structured programs. The polyhedral model is usually imposed in this analysis to compute the communication and data-dependencies analytically. For instance, the work presented in [40] uses

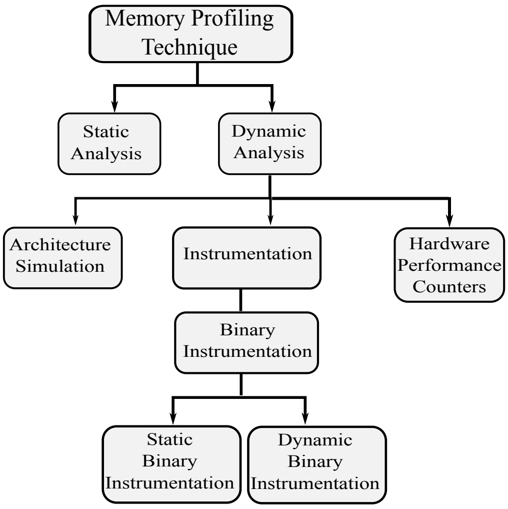
Memory Profiling Technique
Static Analysis
Dynamic Analysis
Architecture Simulation
Instrumentation
Hardware Performance Counters
Binary Instrumentation
Static Binary Instrumentation
Dynamic Binary Instrumentation

Fig. 2. Classification of memory profilers based on profiling technique.

exact data-dependence analysis provided by the polyhedral model to automatically explore the opportunities for communication/computation overlap. This kind of analysis is infeasible for a large number of existing and emerging applications as these programs have irregular structure. Furthermore, problems such as pointer analysis, is still very difficult, even exponential-time algorithms do not always produce sufficiently precise results [41].

Tools based on *dynamic analysis* collect information by running the application in a simulator or on the target platform. These are further classified as architecture simulation and instrumentation. *Architecture simulation* involves modeling a virtual computer system with CPU and memory hierarchy. SimpleScalar [42] is an example in this class, which can simulate various architectures with non-blocking caches, speculative execution, and state-of-the-art branch prediction. A drawback of this technique is that it is computationally intensive, which limit its use to small data inputs. Furthermore, simulation with these small data inputs may not exhibit the realistic memory-access patterns.

Binary instrumentation is a widely used instrumentation technique in which an instrumentation tool injects instrumentation code to the compiled binary. This type of instrumentation can be done statically or dynamically.

Static binary instrumentation was pioneered by ATOM [43]. ATOM organizes the final executable such that the application program and user's analysis routines run in the same address space. Hence, there is a possibility to mix code and data in an executable. Third Degree [44] and Graphical program analysis toolkit [16] by HP are example of tools in this regard.

Dynamic binary instrumentation involves the dynamic compilation of binary of an application to insert the instrumentation code anywhere in it. The program binary is instrumented just before its execution. Examples of tools utilizing this technique are QUAD [21] and PINCOMM [22], [26] which are based on Pin [25]. Similarly, Memcheck [45] and Redux [24] are examples of the tools based on Valgrind [46].

V. Comparison of Data-Communication Profilers

Table III lists the profilers which can be utilized for memory-access optimizations. To present a combined view, this table depicts the classification of these profilers on the all the three aspects of the proposed criteria. An important observation that can be made from this table is that hardware performance counters and dynamic binary instrumentation is the most widely used technique utilized by these profilers. Another observation is that most of the existing tools focus on cache-access optimizations. Similarly, a number of tools exist with perform communication profiling for distributed memory systems where communication is explicit. However, very few tools provide architecture independent data-communication profiling information. Therefore, these tools are studied and there strengths and weaknesses are discussed and compared in this section.

Redux provides the communication information at a fine-granularity of operations. Due to the amount of the details involved, it can only be used for very small toy applications, as discussed by authors.

Utilizing the architecture simulation as used by CETA, is computationally intensive. To give reader an idea, Gem5 [48] achieves a simulation speed of 200 KIPS. With this speed, simulating a single core will take around 8 hrs. Hence, this slow simulation speed limits the use of such tools to small data inputs. Simulation with these small data inputs may not exhibit the realistic memory-access and data-communication patterns. Furthermore, it requires the design and development of a cycle accurate simulator of these architectures as CETAs implementation is necessarily specific to the processor-architecture, simulator, and OS in use, as is also reported by authors. Therefore, our focus in this comparison is limited to QUAD and PINCOMM which are especially designed to provide data-communication information. A summary of various characteristics of both the tools is provided in Table IV.

For the detailed comparison, we performed tests on a $2.66GHz\ Intel(R)\ Core(TM)2\ Quad$ CPU with $12GB$ of main memory. We used Pin $v2.12$ running on Ubuntu 12.04 LTS with Linux kernel $3.5.0-45-generic$.

A. Comparison of the Generated Profiles

Both QUAD and PINCOMM are based on Pin DBI framework, hence are used as pintools. The binary of the application to be profiled is given as an argument to the tool to generate data-communication information.

PINCOMM generates a trace file which is processed by a Perl script to generate user readable information. The advantage of generating this trace file is that temporal aspect of data-communication is preserved. In this way, various phases during the application run can be characterized. The disadvantage is that the size of this trace file grows very large for real applications executed with realistic workloads.

QUAD provides output in the form of a dot graph and XML file. The nodes represent the functions in the application and edges correspond to the data-communication between functions. In each communication relationship, the number

TABLE III
CLASSIFICATION SUMMARY OF MEMORY-ACCESS OPTIMIZATION PROFILERS BASED ON THE PROPOSED CRITERIA.

				ST/MT	Profiling Technique			
					Architecture Simulation	Static Binary Instrumentation	Dynamic Binary Instrumentation	Hardware Counters
Memory-access Optimization Profilers	Cache/ Locality Profilers			ST + MT		HP Graphical Program Analysis Toolkit[16]	Cachegrind[10], HP Caliper[17]	Oprofile[11], Dprof[13], Zoom[14], NUMATop[12] Vampir [30], Intel Vtune[15], AMD CodeXL[18], MS Visual Profiler[19], Oracle Solaris Studio[20]
	Comm. Profilers	Shared Mem.	Arch. Depend	MT	CETA[23]			NUMATop[12], Intel Parallel Studio XE[29], AMD CodeXL[18], Nvidia NVVP[47]
			Arch. Indep.	ST			QUAD[21], Pincomm[22], Redux[24]	
				MT			Pincomm[22]	
		Distributed Memory		MT			TAU[31], mpiP[32], Scalasca[33], periscope[34]	Vampir Toolset[30], TAU[31], Intel Parallel Studio XE[29], Oracle Solaris Studio[20]

TABLE IV
COMPARITIVE SUMMARY OF QUAD AND PINCOMM.

Category	QUAD	Pincomm
Input	Binary	Binary
Input Type	ST	ST/MT
Output	dot, xml	csv
Technique	DBI	DBI
Internal Data Structure	Trie	Hash table
Availability	Open source	Open source
Based on	Intel Pin	Intel Pin
Supported OS	Win, Linux, OS X	Win, Linux, OS X
Supported Architecture	Intel, ARM	Intel, ARM
Reported Metrics	+ (Bytes, UNMA, UNDV)	- (Bytes only)
Profiling Granularity	- (8-bit only)	+ (8,16,32,64-bit)
Execution-time Overhead	+	-
Memory-usage Overhead	-	+
Documentation	+	-

+ indicates profiler is better in this category

TABLE V
OVERHEAD COMPARISON OF QUAD AND PINCOMM

Domain	Application	Execution-time Overhead		Memory-usage Overhead	
		QUAD	Pincomm	QUAD	Pincomm
Img. Proc.	canny	342.8	712.8	1209.3	204
	KLT	1596.5	3580.2	751.3	152.5
SPLASH-2	ocean-NC	2503.1	3774.6	377.7	64.4
	fmm	2100.8	2657.7	340.2	55.9
	raytrace	2897.3	6690.7	361.3	61.3
Bio Inform.	bwa-mem	1693.3	3765	410.5	73.5
	Average	1855.63	3530.17	575.05	101.93

of bytes (Bytes), Unique Memory Addresses (UnMAs) and number of Unique Data Values (UnDVs) are reported on the edges. Furthermore, the intensity of the data-communication is also depicted by the color of the edge in the descending order of Red, Brown, Green etc.

QUAD only supports sequential applications while PINCOMM can also profile multi-threaded applications. Inter-thread data-communication information can be utilized to see the effect of parallelization and drive the mapping of threads to cores for reduced inter-core data-communication.

B. Comparison of Overheads

Both the tools are utilizing dynamic analysis to report the data-communication information, hence large overhead is expected. In order to perform a comparison of the execution-time and memory-usage requirements of QUAD and PINCOMM, we run the two profilers on the same machine to generate the data-communication information. The execution-time is the wall-clock time measured in seconds by the Linux `time` utility. The memory usage is the peak resident set size (`VmHWM`) measured in Mega Bytes (MB) by using the Linux `/proc/<pid>/status`.

Table V details the execution-time and memory-usage overhead of PINCOMM and QUAD. These results are provided for applications from various domains as depicted in the first two columns the table. The numbers in Columns 3 and 4 present the ratios of application execution-time with the native application execution time. These numbers represent the slow-down caused by the application execution because of the profiling. For example, PINCOMM and QUAD slow the execution of *canny* application by a factor of 712.8 and 342.8, respectively. In order to compare the two profilers, Column 5 lists the ratio of the PINCOMM overhead compared to QUAD. Similarly, Columns 6 and 7 report the memory-usage overhead caused by profiling. Column 8 report the ratio of the QUAD memory-usage overhead compared to the PINCOMM.

It can be seen from the results in Table V that, on the average, PINCOMM has about 2 × higher execution-time overhead than QUAD to generate the same information. Main source of the overheads in these tools is the data-structure which stores and retrieves the information about the producer of a memory address. This data-structure is critical to the performance of these tools as it is accessed on each memory read/write performed in the application. In this regard, PINCOMM uses the STL map, whereas QUAD uses a Trie data-structure. Access time increases linearly with the increase in number of accesses for STL map, whereas, in the case of Trie, it stays constant. This means the performance of STL map suffers with the growing size of map, due to the growing application complexity.

Second reason for the variation in overheads of these tools is that PINCOMM writes the gathered information to the disk, whereas QUAD keeps this information in the memory. Therefore, on the average, the memory-usage of QUAD is about 5 × higher than PINCOMM. This is because of the space required for extra metrics reported by QUAD, which are stored in the internal data-structure in the memory, resulting in the higher memory-usage overhead than PINCOMM. In short, QUAD makes a trade-off in order to be time-efficient, by keeping information in the memory, while PINCOMM is space-efficient as it commits the information to the disk.

VI. DISCUSSION AND RECOMMENDATIONS

In this section, we summarize some recommendations for the improvements of existing data-communication profilers and/or the design future ones based on the study in this work.

1) **Reduction of overheads:** Reducing the execution-time and memory-usage is critical for the usability of the tools. This implies the improvement of the following:
   - Efficient trace collection by utilizing hardware performance counters
   - Efficient storage of producer consumer relationship
   - Efficient design to shift computation from analysis to instrumentation.
   - Configurable Profiling granularity
2) **Architecture independent communication characterization:** to support sequential applications as well.
3) **Detection of communication patterns:** Spatial and temporal data-communication information is important. This will also provide insights in mapping the data-structure in an application to the architecture memory hierarchy.
4) **Source-code related profiling information:** In order to make the profile easily usable by programmers. This is also important for the automation of communication-aware optimizations of applications.

VII. Conclusion

Both the memory bottleneck and the multi-core trend create the need for detailed data-communication profiling. In this work, we have discussed various memory profilers, with a deeper focus on data-communication profiling. We have proposed a categorization of memory profilers based on the profiling objective and profiling technique. In addition, we have provided a detailed comparison of the existing data-communication profilers. The important features of these profilers have been extensively discussed. Furthermore, the shortcoming in these tools are highlighted, which serve as recommendations for the improvement of the existing and/or the design of future data-communication profilers.

Acknowledgment

This research is partially supported by the Artemis EMC2 project (grant 621429), the Artemis Almarvi project (grant 621439) and the Artemis Crafters project (grant 295371).

References

[1] M. Horowitz *et al.*, "How scaling will change processor architecture," in *ISSCC*, 2004, pp. 132–133 Vol.1.

[2] Y. Taur, "Cmos design near the limit of scaling," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 213–222, March 2002.

[3] W. A. Wulf *et al.*, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, Mar. 1995.

[4] G. Martin, "Overview of the MPSoC design challenge," in *43rd ACM/IEEE DAC*, 2006, pp. 274–279.

[5] S. Borkar *et al.*, "The future of microprocessors," *Commun. ACM*, vol. 54, pp. 67–77, May 2011.

[6] B. Wun, *Survey of Software Monitoring and Profiling Tools*.

[7] J. Tong *et al.*, "Profiling tools for fpga-based embedded systems: Survey and quantitative comparison," *Journal of Computers*, vol. 3, no. 6, 2008.

[8] S. L. Graham *et al.*, "Gprof: A Call Graph Execution Profiler," *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, 1982.

[9] A. Jaleel *et al.*, *CMP$im: A Pin-Based On-The-Fly Multi-Core Cache Simulator*.

[10] N. Nethercote, "Dynamic binary analysis and instrumentation," Ph.D. dissertation, University of Cambridge, UK, Nov 2004.

[11] W. Cohen, "Multiple Architecture Characterization of the Build Process with OProfile," 2003. URL: http://oprofile.sourceforge.net

[12] Y. Jin, "Numatop: A tool for memory access locality characterization and analysis," https://01.org/numatop, 2013.

[13] A. Pesterev *et al.*, "Locating cache performance bottlenecks using data profiling," ser. EuroSys, 2010, pp. 335–348.

[14] "Zoom by Rotate Right," http://www.rotateright.com/zoom.

[15] "vTune by Intel," http://software.intel.com/en-us/intel-vtune.

[16] "Program Analysis Toolkit by Hewlett Packard," http://h30097.www3.hp.com/developerstoolkit/tools.html.

[17] "Caliper by Hewlett Packard," https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=CALIPEREVAL.

[18] "CodeXL by AMD," http://developer.amd.com/tools-and-sdks/opencl-zone/codexl.

[19] "Visual Profiler by Microsoft," http://msdn.microsoft.com/en-us/library/aa969767%28v=vs.110%29.aspx.

[20] "Solaris Studio by Oracle," http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index-jsp-142272.html.

[21] S. Ostadzadeh, "Quantitative application data flow characterization for heterogeneous multicore architectures," Ph.D. dissertation, TU Delft, Dec 2012.

[22] W. Heirman *et al.*, "A communication profiler to optimize embedded resource usage," *Annual Workshop on Circuits, Systems and Signal Processing*, 2009.

[23] A.-H. Liu *et al.*, "Automatic run-time extraction of communication graphs from multithreaded applications," in *CODES+ISSS*, Oct 2006.

[24] N. Nethercote *et al.*, "Redux: A dynamic dataflow tracer," *Electronic Notes in Theoretical Computer Science*, no. 2, pp. 149–170, Oct. 2003.

[25] C. Luk *et al.*, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *PLDI '05*. New York, NY, USA: ACM, 2005, pp. 190–200.

[26] W. Heirman *et al.*, "PinComm: characterizing intra-application communication for the many-core era," in *ICPADS*, Dec. 2010, pp. 500–507.

[27] P. Magnusson *et al.*, "Efficient memory simulation in simics," in *Simulation Symposium*, Apr 1995, pp. 62–73.

[28] M. Snir *et al.*, *MPI-The Complete Reference, Volume 1: The MPI Core*, 2nd ed. Cambridge, MA, USA: MIT Press, 1998.

[29] "Parallel Studion XE by Intel," https://software.intel.com/en-us/intel-parallel-studio-xe.

[30] A. Knpfer *et al.*, "The vampir performance analysis tool-set," in *Tools for High Performance Computing*, 2008, pp. 139–155.

[31] "TAU Performance System," cs.uoregon.edu/research/tau/home.php.

[32] "mpiP: Lightweight, Scalable MPI Profiling," mpip.sourceforge.net.

[33] M. Geimer *et al.*, "The scalasca performance toolset architecture," *Concurr. Comput. : Pract. Exper.*, vol. 22, no. 6, pp. 702–719, 2010.

[34] M. Gerndt *et al.*, "Automatic performance analysis with periscope," *Concurr. Comput. : Pract. Exper.*, vol. 22, no. 6, pp. 736–748, 2010.

[35] I.-H. Chung *et al.*, "MPI Performance Analysis Tools on Blue Gene/L," in *SC*, Nov 2006.

[36] H. Brunst *et al.*, "Performance analysis of large-scale OpenMP and hybrid MPI/OpenMP applications with vampir NG," in *OpenMP Shared Memory Parallel Programming*, 2008, vol. 4315, pp. 5–14.

[37] S. H. Hung *et al.*, "Trace-based performance analysis framework for heterogeneous multicore systems," ser. ASPDAC '10, 2010, pp. 19–24.

[38] S. H. Hung *et al.*, "New tracing and performance analysis techniques for embedded applications," ser. RTCSA, 2008, pp. 143–152.

[39] M. Bach *et al.*, "Analyzing parallel programs with pin," *Computer*, vol. 43, no. 3, pp. 34–41, Mar. 2010.

[40] S. Pellegrini *et al.*, "Exact dependence analysis for increased communication overlap," ser. LNCS, 2012, vol. 7490, pp. 89–99.

[41] M. D. Ernst, "Static and dynamic analysis: Synergy and duality," in *WODA*, Portland, Oregon, May 2003, pp. 24–27.

[42] T. Austin *et al.*, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb 2002.

[43] A. Srivastava *et al.*, "ATOM: a system for building customized program analysis tools," *SIGPLAN*, vol. 29, no. 6, pp. 196–205, Jun. 1994.

[44] "Third Degree by Hewlett Packard," http://h30097.www3.hp.com/developerstoolkit/tools.html.

[45] J. Seward *et al.*, "Using valgrind to detect undefined value errors with bit-precision," in *USENIX ATC*, ser. ATEC '05, 2005.

[46] N. Nethercote *et al.*, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *SIGPLAN*, vol. 42, no. 6, pp. 89–100, Jun. 2007.

[47] Nvidia, "Nvprof and nvvp, nvidia command-line and visual profilers." URL: http://docs.nvidia.com/cuda/profiler-users-guide

[48] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.