2013 16th Euromicro Conference on Digital System Design

The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices

Marco Solinas[1], Rosa M. Badia[2], François Bodin[3], Albert Cohen[4], Paraskevas Evripidou[5], Paolo Faraboschi[6], Bernhard Fechner[7], Guang R. Gao[8], Arne Garbade[7], Sylvain Girbal[9], Daniel Goodman[10], Behran Khan[10], Souad Koliai[8], Feng Li[4], Mikel Luján[10], Laurent Morin[3], Avi Mendelson[11], Nacho Navarro[2], Antoniu Pop[4], Pedro Trancoso[5], Theo Ungerer[7], Mateo Valero[2], Sebastian Weis[7], Ian Watson[10], Stéphane Zuckermann[8], and Roberto Giorgi[1]

[1] Dip. di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Italy. {solinas, giorgi}@dii.unisi.it
[2] Barcelona Supercomputing Center, Spain. {rosa.m.badia, nacho, mateo.valero}@bsc.es
[3] CAPS Enterprise, France. {francois.bodin, laurent.morin}@caps-entreprise.com
[4] INRIA, France. {albert.cohen, antoniu.pop}@inria.fr
[5] University of Cyprus, Cyprus. {pedro, skevos}@cs.ucy.ac.cy
[6] Intelligent Infrastructure Lab, Hewlett Packard, Barcelona, Spain. paolo.faraboschi@hp.com
[7] University of Augsburg, Germany. {bernhard.fechner, arne.garbade, theo.ungerer, sebastian.weis}@informatik.uni-augsburg.de
[8] University of Delaware, Delaware, USA. {szuckerm, koliai}@eecis.udel.edu, ggao@capsl.udel.edu
[9] THALES, France. sylvain.girbal@thalesgroup.com
[10] University of Manchester, United Kingdom. mikel.lujan@manchester.ac.uk, watson@cs.man.ac.uk
[11] Technion, Israel. avi.mendelson@tce.technion.ac.il

***Abstract*— Thanks to the improvements in semiconductor technologies, extreme-scale systems such as teradevices (i.e., composed by 1000 billion of transistors) will enable systems with 1000+ general purpose cores per chip, probably by 2020. Three major challenges have been identified: programmability, manageable architecture design, and reliability. TERAFLUX is a Future and Emerging Technology (FET) large-scale project funded by the European Union, which addresses such challenges at once by leveraging the dataflow principles. This paper describes the project and provides an overview of the research carried out by the TERAFLUX consortium.**

Keywords—TERAFLUX; dataflow; programming model; compilation; reliability; architecture; simulation; many-cores; exascale computing; multi-cores.

I. INTRODUCTION

Continuous improvements in silicon manufacturing technology, such as FinFET [1] transistors and 3D-die stacking [2], start showing a tremendous impact on the near future computer systems. New silicon technology generations continue to double the number of transistors in every generation, but with no significant frequency enhancements and with the cost of power density. These facts open the doors for new chips (that we call *teradevices*) with a huge number of transistors (for current ITRS [3] projections, 1 Tera or 10^{12} transistors) with the possibility of exploiting the large amount of parallelism in different ways.

In future exascale machines, the number of general purpose cores (i.e., compute elements) per die will exceed those of current systems by far. This suggests a major change in the software layers that are responsible of using all such cores. The three major challenges regard programmability, reliability and complexity of design. Also, a new Program eXecution Model (PXM) [6][49][12] seems suited in order to address such challenges.

Given the large number of transistors and the diversity in the requirements for different applications, it is natural to expect that these massively parallel (or concurrent tera-device) systems will be composed of heterogeneous cores. Thus, programmability of such large-scale systems will be a major challenge. Moreover, such large systems are expected to become more and more susceptible to failures, due to the increasing sensibility to process variations and manufacturing defects. Thus, this extreme scale of device integration represents a second major concern, in terms of reliability, for future many-core systems. Finally, the software industry is lagging behind as general purpose applications cannot take advantage of more than a handful a number of cores compared to the larger degree of parallelism offered by the current and future processors. Starting from this premise, there is the need for new ways to exploit the large parallelism offered by future architectures as expected to be a reality beyond the year 2020.

The dataflow concept is known to overcome the limitations of the traditional control-flow model by exploring the maximum parallelism and reducing the synchronization overhead. As recalled by Jack Dennis [4], dataflow is "*A Scheme of Computation in which an activity is initiated by presence of the data it needs to perform its function*". The dataflow paradigm is not new, but recently it has met mature silicon technology and architectural models to take advantage from the large intrinsic parallelism.

TERAFLUX [43] is a Future Emerging Technologies (FET) large-scale project funded by the European Union. The aim is to exploit the dataflow paradigm in order to address the three major challenges presented above (i.e., programmability, reliability, and manageable architecture design). Since we are targeting 1000+ core systems, the dataflow paradigm enables us to use the increased degree of parallelism which emerges in future teradevices.

DOI 10.1109/DSD.2013.39

The rest of the paper is organized as follows. Section II provides a general overview of the project. Remaining sections are focused on describing the concepts together with the major achievements resulting from our research activity. In particular, Section III describes the possible applications based on the OmpSs programming model, while Section IV details a further possibility of using a productivity language such as Scala thanks to a dataflow runtime called DFScala. Another common layer (OpenStream, presented in Section V) is used for mapping feed-forward dataflow into lower-level dataflow threads as expressed by the T* Instruction Set Extension, described in Section VI, together with the architecture of our target system. Section VII describes the Fault Detection Units (FDUs), which provide fault management and fault detection through monitoring techniques and redundant execution of dataflow threads. The experiments are integrated into a common simulator based on the HPLabs COTSon [5], presented in Section VIII. Finally, Section IX introduces the codelet model, while Section X concludes the paper.

II. General Overview of TERAFLUX

To investigate our concepts, we use dataflow principles at any level of a complete transformation hierarchy, starting from general and complex applications able to load properly at teradevice through programming model, compilation tools, reliability techniques and architecture. Fig. 1 shows the TERAFLUX layered approach.

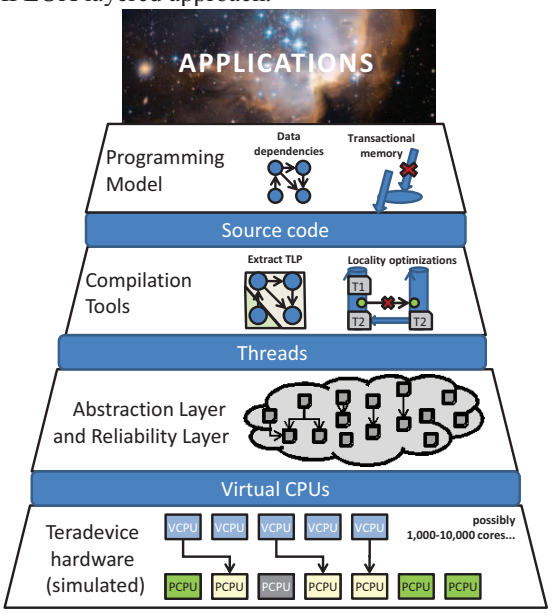


Fig. 1 The TERAFLUX Transformation Hierarchy.

Different layers allow to transform applications source code into a dataflow-style binary, and to execute it on the target architecture. The top level of this hierarchy is represented by real world applications, which allow us to stress the underlying teradevice hardware. In the TERAFLUX project, implicit parallelism refers to the set of constraints on concurrent execution of threads, and the expression of these constraints in the source code. These constraints can be dependences, atomic transactions, synchronization barriers, privatization attributes, memory layout and extent properties, and a wide variety of hints. An explicitly parallel program, on the other hand, is made of concurrency constructs making the thread creation, termination, and possibly some target-specific aspects of the execution explicit [22][23][24].

A dataflow oriented programming model allows expressing data dependencies among the concurrent tasks of an application. In the dataflow terminology, such concurrent tasks are called *dataflow threads*, or simply threads when clear from the context. Nevertheless, applications use large data structures with in-place updates, for efficient memory management [18][19][52] and copy avoidance. Such applications require a mechanism to express the non-interference of concurrent updates to shared data. To meet such need, we selected Transactional Memory (TM), as the most promising programming construct and concurrency mechanism for specifying more general forms of synchronization among threads, while preserving the composability of parallel dataflow programs and promising a high level of scalability [38]. We achieve this by defining a specific layer for studying the integration between the TM and dataflow programming models [25][28][54].

Besides the programming model, implicit parallelism must be exploited by a compilation tool-chain [40][57][58], being able to convert dependences and transactions, into scalable target-specific parallelism. It is also responsible for properly managing the inter-node communications and a novel memory model. Compiler effectiveness is guaranteed by the implementation of a generalization of the state of the art algorithms to expose fine-grained dataflow threads from task-parallel OpenMP-, StarSs- or HMPP-annotated [61] programs. The algorithm generalization leverages a new dependence-removal technique to avoid the artificial synchronizations induced by in-place updates in the source program [34][51].

Our goals in designing an efficient compilation tool-chain are to capture the important data reuse patterns, to optimize locality and reduce communication bandwidth, and to provide compiler support for transaction semantics embedded into a dataflow programming model. Both productivity and efficiency programming layers are supported. Compiler directives are used to lower the abstraction penalty of the productivity layer, and to exploit parallelism and locality explicitly.

As mentioned in the Section I reliability will be a major concern for future many-cores architectures. With the aim of limiting the impact of faults in the target architecture, dedicated hardware modules are devoted to monitor the healthiness of the system, and drive specific counteractive measures [29][30]. To achieve this goal in TERAFLUX, we focused on inter-core fault detection techniques using Fault Detection Units (FDUs). We considered different FDU variants (push, pull, alert mechanisms for heartbeat messages), FDU implementations, and interfaces.

The design space exploration finally resulted in a proposal of a functional FDU specification based on the MAPE (Monitoring, Analysis, Planning, and Execution) [42] cycle of Organic Computing. Abstract message interfaces of the FDU to all communication units (e.g., FDU-core, FDU-operating system, etc.) were specified for push, pull, and

alert messages. Core healthiness is monitored by exploiting currently available performance monitoring and machine check hardware features (e.g., machine check architecture of current AMD/Intel processor families).

System resources are managed at the highest level by the operating system. The main objective of the operating system is to balance the workload among the nodes while keeping an acceptable level of fault tolerance. The control of scheduling and the resource managing are hierarchically performed: distributed FDUs are used to guarantee the characteristics of the basic nodes by accessing the different resources such as the cores, and local memories.

Similarly to the FDU, the other resources of the TERAFLUX system are hierarchically organized, mainly resembled to a set of nodes interconnected with each other. Each node contains a hardware structure for scheduling the medium/fine-grain dataflow threads generated by the compilation tool-chain, and execute them. The TERAFLUX architecture is designed in order to support the programming and execution models developed by the higher level layers. At this point the project focuses on defining the basic architecture modules as well as the necessary instruction extensions to support the programming and execution model. The basic architecture consists of a number of multi-core nodes. We are ISA agnostic, in principle, but we wanted to demonstrate our concept with a well-known ISA such as the x86-64. The nodes are interconnected through a Network on Chip (NoC). TERAFLUX supports a global address space across the whole system. Each node contains a portion of the global memory. There is no need for traditional coherency because the dataflow model is based on the single assignment semantics. Different memory types (e.g. shared and non-shared) are defined as to store particular data and metadata of the programs. Finally, there are custom hardware modules that are added to the cores and nodes to actively support dataflow thread scheduling and reliability monitoring (TSUs or Thread Scheduling Units). An abstraction software layer is defined to allow interfacing the operating system and the underlying hardware.

The aim of the lowest layer of the TERAFLUX hierarchical approach is to the provide software and hardware infrastructures capable of simulating all the modules composing the target dataflow system. In TERAFLUX we rely on a state of the art many-core simulation infrastructure (HPLabs COTSon [5]), which is able to scale up the number of cores to two orders of magnitudes larger than what is currently available. This simulation infrastructure represents a common point for all the partners, allowing them to test their research ideas and integrating them in a common platform.

III. PROGRAMMING DATAFLOW ARCHITECTURES WITH TASK-BASED APPROACH

One of the key aspects of the TERAFLUX project is the proposal of a new programming and execution model [32][33][45] based on dataflow instead of traditional control-flow. Dataflow is known to overcome the limitations of the traditional control-flow model by exploring the maximum parallelism and reducing the synchronization overhead. The StarSs programming model [8] provides a paradigm for the development of applications following the sequential programming paradigm but based on an execution model that exploits the inherent concurrency of the applications taking into account the existing data dependencies.

The code is developed in a standard, sequential language, such as C or Fortran. On the user's side there are no explicit parallel constructs, like in thread or stream models.

```
#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]T) input ([TS][TS]B)
void strsm (float *T, float *B);
#pragma omp task input ([TS][TS]A,[TS][TS]B)\
 inout ([TS][TS]C )
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]A)\
 inout ([TS][TS]C)
void ssyrk (float *A, float *C);


void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
      }
   }
}
```

Fig. 2 Example of code annotated with OmpSs compiler directive.

Since the paradigm is task-based, the programmer needs to add annotations or compiler directives to the code to mark those pieces of code which are to be considered a task and the directionality of key arguments of the tasks. At runtime, this information about the directionality of the task data is used to build a task data-dependence graph that exhibits the inherent data dependences of the application as well as its potential task parallelism. Current efforts at the Barcelona Supercomputing Center are focused on the OmpSs [9][46] implementation of StarSs which extends the OpenMP explicit tasks [10] with dependence clauses that indicate the directionality of the tasks arguments, and on the Task Superscalar design [59][60].

Fig. 2 shows an example of OmpSs code. The example implements a Cholesky factorization. The kernels of the factorization have been annotated with OmpSs compiler directives. The directionality clauses (*input*, *output*, *inout*) indicate whether the given parameter is read, write or read and write in the scope of the task.

In the framework of TERAFLUX, OmpSs has been used as a high level programming model to develop applications. The OmpSs coarse-grained tasks are then translated to finer dataflow threads that are executed in the dataflow architecture. OmpSs is available as open source and can be downloaded from *http://pm.bsc.es/ompss*.

IV. DFSCALA: CONSTUCTING AND EXECUTING DATAFLOW GRAPHS

One part of this project is the construction of a high level dataflow framework which serves two purposes: *i*) to provide a high productivity language in which to construct dataflow

programs, and *ii*) to provide a high level platform for experimenting with new ideas such as using the type system to enforce different properties of the dataflow graph and different memory models. DFScala provides a key foundation and implements the base functionality of this research platform. One distinguishing feature of DFScala is the static checking of the dynamically constructed DF graph.

In a dataflow program the computation is split into sections. Depending on the granularity of the program these vary from a single instruction to whole functions which can include calls to other functions, allowing arbitrarily large computation units. All of these sections are deterministic based on their input and side-effect free. The execution of the program is then orchestrated through the construction of a directed acyclic graph where the nodes are the sections of computation and the vertices are the data dependences between these. An example of this can be seen in Fig. 3. Once all the inputs of a node in the graph have been computed the node can be scheduled for execution.

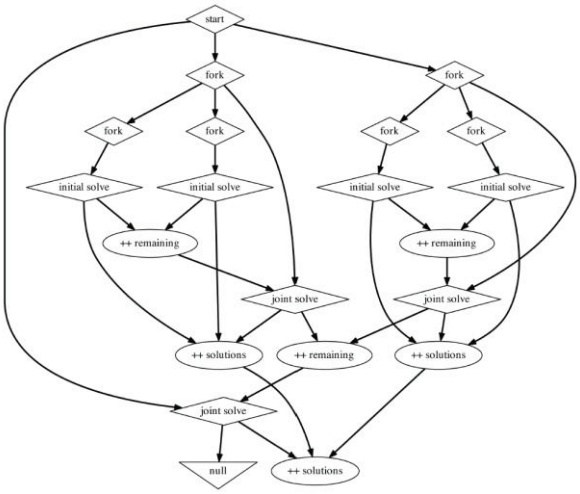


Fig. 3 An instance of a dataflow graph for a circuit routing algorithm.

The DFScala library is open source and provides the functionality to construct and execute DF graphs in Scala. The nodes in the graph are dynamically constructed over the course of a program and each node executes a function which is passed as an argument. The arcs between nodes are all statically typed. More details are in recent work [17][26].

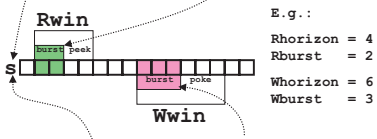
V. The OpenStream extension to OpenMP

A key point of TERAFLUX is the compilation flow, that has been vastly remodeled to target the reference architecture. In particular, such compilation flow has been implemented as a front- and middle-end extension to GCC 4.7.1. Starting from a programming model which extends OpenMP to support streaming task directives, called OpenStream [35][20][50], designed by INRIA, the compiler is able to expand streaming task directives into dataflow threads and point-to-point communications. Programs written in higher level languages such as StarSs can be translated source-to-source to OpenStream using slightly modified implementations of their dependence resolver. The rationale for designing such streaming extension is motivated by the need to capture dataflow dependences explicitly in a parallel language, by the quest for increased productivity in parallel programming, and by the strong evidence that has been gathered on the importance of pipeline parallelism for scalability and efficiency.

```
input/output (list)
   list   ::= list, item
              item
   item   ::= stream
              stream >> window
              stream << window
   stream ::= var
              array[expr]
   expr   ::= var
              value
```

(a)

```
int s, Rwin[Rhorizon];
int Wwin[Whorizon];
input (s >> Rwin[Rburst])
```




```
output (s << Wwin[Wburst])
```

(b)

Fig. 4 Syntax for input and output clauses (a) and illustration of stream access through windows (b).

OpenStream provides the programmer with the possibility of specifying the flow of data between OpenMP tasks, and to build the program task graph, via simple annotations. OpenStream allows dynamically constructed task graphs, as well as multiple tasks interleaving their communications in the same streams, and arbitrary and variable fan-in, fan-out, and communication rates. This approach differs from state-of-the-art streaming languages, which mostly rely on static and/or regular task graphs.

The OpenStream compiler is responsible for converting the rich dependence patterns among dynamically constructed tasks into low-level, feed-forward dataflow operations expressed in the T* instruction set. This conversion involves streams as an intermediate data structures for producer threads to discover their consumers.

The OpenStream syntactic extension to the OpenMP language specification consists of two additional clauses for *task* constructs, the *input* and *output* clauses, both taking a list of items, describing the stream and its behaviours. For example, within the body of a task, one can need to access each element of the stream one at a time (hence, the stream abbreviated form can be adopted), or multiple elements at a time through sliding windows (the forms adopting the << and >> stream operators are the most suitable). Fig. 4 shows both the syntax of the additional clauses (a) and an example of stream accessed via sliding window (b).

OpenStream derived from the results of the ACOTES FP6 project. It is also adapted to embedded systems in the PHARAON FP7 project, to support dynamic voltage and frequency scaling under real-time constrains.

VI. THE TERAFLUX REFERENCE ARCHITECTURE

Within the research performed in our project, an important aspect is represented by the execution model and architecture framework [44] including hardware modules to support the execution model. The proposed template for the TERAFLUX architecture is shown in Fig. 5.

The execution model (PXM) is a combination of fine- and coarse-grain threaded DF models including UNISI DTA [6]8, UCY DDM [7] and BSC StarSs [8]. In addition, the transactional support is added to the dataflow model, which allows covering more applications: those that include dataflow threads that modify shared state. Combining dataflow with transactions is a unique feature of this project.

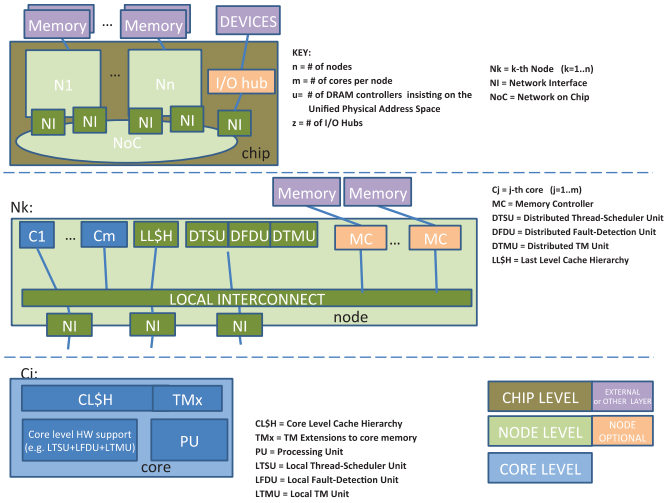


Fig. 5 TERAFLUX Architecture Template.

Current work is towards finalizing different parts of the system so that we can achieve the following main goals: *i*) Improving the overall performance (e.g., measured as execution time) by a factor of two compared to the baseline architecture (1000 complex cores); *ii*) Reducing the overall needed size of caches by 50% without any performance loss compared to the same baseline architecture.

DDM. Data-Driven Multithreading (DDM) is one of the dataflow models studied in TERAFLUX. DDM is a multithreaded model that applies Dynamic Dataflow principles for communication among threads and exploits highly efficient control-flow execution within a thread. The core of DDM is the Thread Scheduling Unit that provides the Data-Driven scheduling of the Threads. DDM does not need traditional memory coherence because it enforces the single assignment semantics for data exchange among threads. Furthermore, it employs prefetching of input data before a thread is scheduled for execution by the TSU. DDM prefetching is deterministic and can be close to optimal because the TSU knows at any time which threads can be executed on which core and thus can initiate the necessary prefetching. DDM based processors can achieve high performance with simpler designs, as they do not need complex and expensive modules like out-of-order execution.

Recently we have investigated whether DDM can achieve high performance in HPC applications [39]. We have implemented three linear algebra applications (Matrix Multiplication, LU decomposition and Cholesky decomposition) using DDM and tested their scalability for a large core number. The results so far [39] are encouraging and show that DDM can handle the parallelization required for linear algebra applications for present and future multi- and many-core systems. Thus, DDM should be further investigated as viable candidate for HPC.

T (T-STAR) INSTRUCTIONS.* To support the execution of DF-Threads, we designed a minimalistic extension of the x86-64 ISA, that we call T-Star (or T-*) [31]. The key-points of this ISE are: *i*) it enables an asynchronous execution of threads, that will execute not under the control-flow of the program but under the dataflow of it; *ii*) the execution of a DF-thread is decided by an core-external component that we call DTS (or Distributed Thread Scheduler); *iii*) the types of memory that are used are distinguished in 4 main types (1-to-1 communication or Thread Local Storage, N-to-1 or Frame Memory, 1-to-N or Owner Writable Memory, and N-to-N or Transactional Memory).

VII. IMPROVING RELIABILITY BY LEVERAGING DATAFLOW PROPERTIES

The tera-scale level transistor integration capacity of future devices will make them orders of magnitude more vulnerable to faults. Without including mechanisms that dynamically detect and mask faults, such devices will suffer from uneconomic high failure rates. We focus on reliability aspects on four levels within the TERAFLUX architecture, to assemble a reliable system out of unreliable components. These levels are *i*) the cores, *ii*) the nodes, *iii*) the interconnection network, and *iv*) the operating system.

At core and node level, we design specific units responsible for *i*) monitoring the *health state* of the cores, and *ii*) providing information to the hardware scheduler about the detected faults. We call such units *Distributed Fault Detection Unit* (D-FDU, operating at node-level) and *Local Fault Detection Unit* (L-FDU, at core-level). In TERAFLUX, the various D-FDUs detect faults by means of the Double Execution mechanism [41][42][56], a redundant execution scheme for DF-threads that we designed by leveraging the side-effect-free semantic of the dataflow execution. In particular, our mechanism duplicates the execution of each DF-Thread, and compares the results of both executions to check for correctness: L-TSUs are responsible for calculating a CRC-32 signature of both *write sets*, which will be sent to the D-FDU when the thread terminates. If the two signatures differ, a faulty execution is assumed, and the D-TSU is notified. Consequently, the results of the computation are discarded and the DF-threads are restarted on different cores. Fig. 6 compares the dependency graph of a regular dataflow execution to the Double Execution one.

On the interconnection level, efficient methods have been designed to localize faults within the network (router and link) [21][37][55]. The localization technique utilizes the knowledge of the existing heartbeat messages and extracts inherent information from them. Finally, the D-FDUs forward the gathered node health states to the operating system to provide additional information for global scheduling decisions.

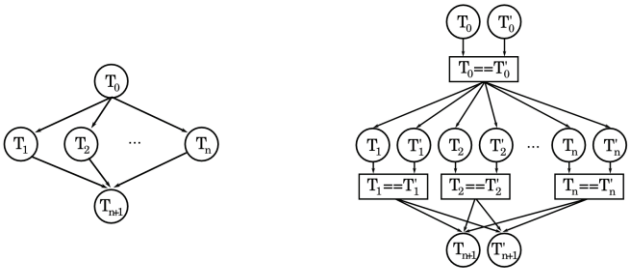


Fig. 6 Dependency graph for regular dataflow execution (left graph) and double execution (right graph).

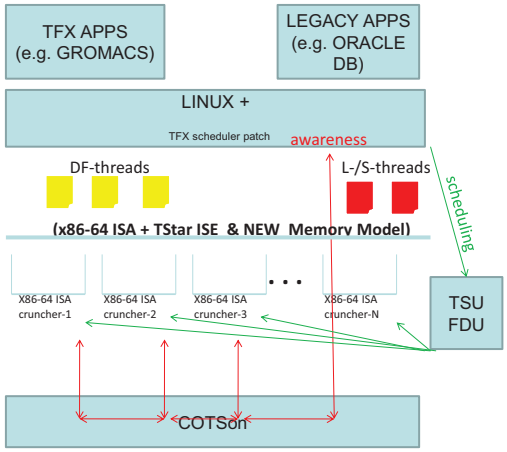


Fig. 7 Overall picture of the "simulator illusion". A number n of VCPUs can be used as "workers" or "x86-64 cruncher" or "auxiliary cores"; a generic kth VCPU can be used as service core. L- and S-threads represent legacy and system Threads, that our system is able to execute.

VIII. The Common Evaluation Platform

The TERAFLUX project relies on a common evaluation platform [36] that is used by the partners with two purposes: *i*) evaluate and share their research by using such integrated, common platform, and *ii*) transfer to the other partners the reciprocal knowledge of such platform.

The common platform includes not only the COTSon simulator, but it also encompasses the compiler that is being developed in the project, as well as other useful tools (e.g., McPAT [11] for power estimation) that we integrated to meet our research needs. Nevertheless, the COTSon simulator is the main component of the overall platform.

COTSon has been chosen because since the first month of the project it could potentially provide an instance of a Teradevice system. An overall picture that highlights what the software sees - as a simulated machine - is shown in Fig. 7. The relevant point is that the software should not look inside the COTSon simulated machine, or make any assumption about the developing Architecture: the exact purpose is to decouple the process of software design and hardware design (while keeping a "contract" between them)[47][48].

Therefore the simulation software exposes a number of virtual processors where the guest software can run unmodified (we called them VCPUs). From the software point of view all these VCPUs could be both considered as full x86-64 virtual machines or as simple "x86-64 ISA crunchers" (or "auxiliary cores"). It will be up to the simulator to expose or not the latter capability, but again for the sake of generality the application software should not presume the availability of any OS-service: each VCPU is just a bare machine. The COTSon, on the other side, can implement any virtualization trick to make this illusion becoming available. We can also assume, as a first instance, that one of the VCPUs is the service core(s) (e.g., the k-th) and runs a guest Linux OS that provides the necessary support to load both TERAFLUX Applications (TFX APPS for short) and LEGACY APPS (such as the ORACLE DBMS). This one OS will be internally modified to distribute the DF/L/S-threads in such a way that the TSU is informed.

During these three years of the project, we extended the platform in order to support the TERAFLUX dataflow execution models (both DDM and T*). In particular, we added the full support for the T* extension to the x86-86 ISA, by implementing a model for the TSU. We also added a fault-injection model for evaluating the overhead introduced by the Double Execution mechanism, which was also modeled with the FDU. Finally, we extended the platform in order to pass from a cluster-based view of the target tera-device system, to a many-nodes-per-chip one, by realizing a communication mechanism via the host shared memory, considering appropriate timing model [27].

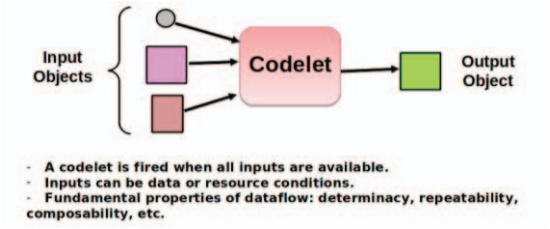


Fig. 8 The Codelet: a fine-grain piece of computation.

IX. The Codelet Model

The codelet model [12] is a hybrid Von Neumann dataflow program execution model (PXM) aimed at providing a fine-grain parallel execution model. It relies on explicit data dependence specified between its units of computations, called codelets. A codelet (Fig. 8) is the main scheduling quanta of the model. The codelets are expected to be generated by a compiler. Previous experience with the codelet model's ancestor, EARTH [13], have shown that automatic partitioning programs into thread that follow dataflow semantics is indeed possible [14].

The codelet PXM relies on an Abstract Machine Model (AMM). It features a hierarchical topology, as well as a heterogeneous architecture. While from a high-level point of view the codelet AMM may not look much different than other machines and it is compatible with the TERAFLUX architectural template, the important part is located at the chip level: the scheduling part is delegated to a dedicated unit (the SU), while the computational part is performed by the computation units (CUs). A codelet is a self-contained sequence of machine instructions scheduled atomically on a computation unit. As a principal scheduling quantum, a codelet, once allocated and scheduled to a computation unit (e.g. a computation core), will keep the computation unit

usefully busy, and will not be pre-empted in most common cases. One feature of the codelet execution is the efficient support of a non-preemptive thread model. A core on which a codelet is running is simply made idle when the codelet is suspended. Under a many-core architecture with hundreds or thousands cores, thread context switching can be costly.

The codelet PXM also relies on a memory model. The codelet memory model is based on Location Consistency (LC) [15]. LC does not require a global ordering of memory operations on the same memory location visible to all processors. Consequently, a memory model based on LC should provide better scalability than other existing cache-coherent based models and protocols.

In the context of the TERAFLUX project, our goal is to study the impact of the codelet runtime DARTS (C++ implementation of the Codelet PXM), on teradevices. First, a comparison study has been done between the Codelet PXM and DF-Thread, the execution model on which COTSon (i.e., the TERAFLUX's simulation platform) is based. Then, we ported the DARTS runtime on COTSon taking advantage of the features provided by the simulator (such as the TSUs) and implicitly by the DF-Thread execution model in COTSon. We are also studying different percolation techniques for teradevices. Percolation [16] is a mechanism that determines how code and/or data should be located, and where, on a given machine. Its goal is to guarantee as much locality as possible for the on-going computation.

Other projects seems to move along similar directions [62][63].

X. CONCLUSIONS

We presented TERAFLUX, a Future Emerging Technology Large-Scale Project funded by EU, which is at the forefront of major research challenges such as programmability, manageable architecture design reliability of many-core or 1000+ cores chip. We described the project transformation hierarchy, and provided a description of the major achievements coming from the research activities carried out in the project. Such achievements encompass applications and programming models for large-scale systems exploiting the dataflow principles, the compiler, fault-tolerance techniques, the architecture and the simulation infrastructure needed for the evaluation of the proposed research.

ACKNOWLEDGMENTS

This work was partly funded by the European FP7 project TERAFLUX (id. 249013) http://www.teraflux.eu. Prof. Avi Mendelson's work has been carried out at Microsoft R&D, Israel.

REFERENCES

[1] M. Jurczak, N. Collaert, A. Veloso, T. Hoffmann, S. Biesemans, "Review of FINFET technology", Proc. of the 2009 IEEE Intern.l Silicon on Insulator (SOI) Conference, 5-8 Oct. 2009, pp. 1-4

[2] R. Chanchani, "3D Integration Technologies - An Overview". In *Materials for Advanced Packaging*, Springer 2009

[3] http://www.itrs.net/Links/2011ITRS/Home2011.htm

[4] J. Dennis, "The Data Flow concept Past, Present and Future", Dataflow Execution Models for Extreme Exascale Computing (DFM), Oct. 2011

[5] E. Argollo et al. Cotson infrastructure for full system simulation. Operating Systems Rev, 43:52–61, 2009.

[6] R. Giorgi, Z. Popovic, N. Puzovic, "DTA-C: A Decoupled multi-Threaded Architecture for CMP System. Computer Architecture and High Performance Computing". In Proc. of the 19th Int.l Symp. on Computer Architecture and High-Performance Computing (SBAC-PAD 2007), pp.263-270, October 2007

[7] C. Kyriacou, P. Evripidou, and P. Trancoso, "Data-Driven Multithreading Using Conventional Microprocessors," IEEE Trans. Parallel Distrib. Syst., vol. 17, no. 10, pp. 1176–1188, 2006

[8] J. Planas, R. M. Badia, E. Ayguadé and J. Labarta. "Hierarchical task based programming with StarSs". Int.l Journal of High Performance Computing Applications, vol. 23, no. 3, 284 - 299, Aug. 2009

[9] A. Duran, R. Ferrer, E. Ayguadé, R. M. Badia and J. Labarta, "A Proposal to Extend the OpenMP Tasking Model with Dependent Tasks". Int.l J. of Parallel Programming, v. 37, 3(2009), pp. 292-305.

[10] OpenMP Architecture Review Board. OpenMP 3.1 Specification. http://www.openmp.org, July 2011.

[11] S. Li, J.-Ho Ho Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, N. P. Jouppi, McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. 42nd Annual IEEE/ACM Int.l Symp. on Microarchitecture, 2009. MICRO-42. 12-16 Dec. 2009, pp. 469 – 480.

[12] S. Zuckerman, J. Suetterlein, R. Knauerhase, and G. R. Gao. Using a "codelet" program execution model for exascale machines: position paper. In Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, EXADAPT '11, pages 64–69, New York, NY, USA, 2011. ACM.

[13] H. H. J. Hum, O. Maquelin, K. B. Theobald, X. Tian, G. R. Gao, and L. J. Hendren. A study of the earth-manna multithreaded system. Int. J. Parallel Program., 24(4):319–348, August 1996.

[14] L. Hendren, X. Tang, Y. Zhu, G. Gao, X. Xue, H. Cai, and P. Ouellet. Compiling c for the earth multithreaded architecture. In Int.l Journal of Parallel Programming, pp. 12–23. IEEE Comp. Soc. Press, 1996.

[15] G. R. Gao and V. Sarkar. Location consistency-a new memory model and cache consistency protocol. IEEE Trans. Comput., 49:798–813, August 2000.

[16] G. Gao, K. Likharev, P. Messina, and T. Sterling. Hybrid technology multithreaded architecture. In Frontiers of Massively Parallel Computing, 1996. Proceedings Frontiers '96., Sixth Symposium on the, pages 98 –105, oct 1996.

[17] D. Goodman, B. Khan, S. Khan, M. Luján, I. Watson: Software transactional memories for Scala. J. Parallel Distrib. Comput. (JPDC) 73(2):150-163 (2013)

[18] N. Minh Lê, A. Pop, A. Cohen, and F. Zappa Nardelli. Correct and efficient work-stealing for weak memory models. In Symp. on Principles and Practice of Parallel Programming (PPoPP), Shenzhen, China, February 2013

[19] B. Diouf, C. Hantaş, A. Cohen, Ö. Özturk, and J. Palsberg. A decoupled local memory allocator. ACM Transactions on Architecture and Code Optimization (TACO), selected for presentation at the HiPEAC 2013 Conf., January 2013

[20] A. Pop and A. Cohen. OpenStream: Expressiveness and dataflow compilation of OpenMP streaming programs. ACM Transactions on Architecture and Code Optimization (TACO), selected for presentation at the HiPEAC 2013 Conf., January 2013

[21] A. Garbade, S. Weis, S. Schlingmann, B. Fechner and T. Ungerer, "Impact of Message-Based Fault Detectors on a Network on Chip," in 21th International Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP), Belfast, 2013

[22] J. Ciesko, J. Bueno, N. Puzovic, A. Ramirez, R. M. Badia, J. Labarta, "Programmable and Scalable Reductions on Clusters", IPDPS, 2013

[23] J. Planas, R. M. Badia, E. Ayguade, J. Labarta, "Self-Adaptive OmpSs Tasks in Heterogeneous Environments", IPDPS, 2013

[24] F. Yazdanpanah, D. Jimenez-Gonzalez, C. Alvarez-Martinez, Y. Etsion, R. M. Badia, "FPGA-Based Prototype of the Task Superscalar Architecture", WRC, 2013

[25] I. Herath, D. Rosas-Ham, D. Goodman, M. Luján, I. Watson. A case for Exiting a Transaction in the Context of Hardware Transactional Memory. 7th ACM SIGPLAN Ws on Transact. Computing, 2012.

[26] D. Goodman, S. Khan, C. Seaton, Y. Guskov, B. Khan, M. Lujan, I. Watson, "DFScala: High Level Dataflow Support for Scala", DFM workshop, Minneapolis, USA, September 2012

[27] J. Navaridas, B. Khan, S. Khan, P. Faraboschi, M. Lujan, "Reservation-based Network-on-Chip Timing Models for Large-scale Architectural Simulation," Networks on Chip (NoCS), 2012 Sixth IEEE/ACM Int.l Symposium on , vol., no., pp.91-98, 9-11 May 2012.

[28] R. Gayatri, R. M. Badia, E. Ayguade, M.l Lujan, I. Watson, "Transactional access to shared memory in StarSs, a task based programming model". In proc. of Int.l European Conf. on Parallel and Distributed Computing, EURO-PAR 2012, Rhodes Island, Greece, August 27-31, 2012.

[29] J. Wolf, B. Fechner and T. Ungerer, "Fault Coverage of a Timing and Control Flow Checker for Hard Real-Time Systems," in 18th IEEE International On-Line Testing Symposium (IOLTS '12), 2012.

[30] J. Wolf, B. Fechner, S. Uhrig and T. Ungerer, "Fine-Grained Timing and Control Flow Error Checking for Hard Real-Time Task Execution," in 7th IEEE Symp. on Industrial Embedded Systems (SIES '12), 2012.

[31] R. Giorgi, "TERAFLUX: Exploiting Dataflow Parallelism in Teradevi-ces", ACM Computing Frontiers, , May 2012, pp.303-304.

[32] J. Bueno, J. Planas, A. Duran, R. M. Badia, X. Martorell, E. Ayguade, J. Labarta, "Productive Programming of GPU Clusters with OmpSs". In Proc. of 26th Int.l Parallel and Distributed Processing Symp. (IPDPS), Shanghai, China, May 21-25 2012.

[33] V. Krishnan Elangovan, R.M. Badia, E. Ayguade, "OmpSs-OpenCL Programming Model for Heterogeneous Systems", LCPC, 2012

[34] F. Li, A. Pop, A. Cohen, "Automatic Extraction of Coarse-Grained Dataflow Threads from Imperative Programs", IEEE-Micro.

[35] A. Pop and A. Cohen. Work-streaming compilation of futures. In 5th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), March 2012

[36] A. Portero, A. Scionti, Z. Yu , P. Faraboschi, C. Concatto, L. Carro, A. Garbade, S. Weis, T. Ungerer, R. Giorgi, "Simulating the Future kilo-x86-64 core Processors and their Infrastructure", (SpringSim'12), March 26 - 29, 2012; Orlando, FL, USA.

[37] B. Fechner, A. Garbade, S. Weis, T. Ungerer, Title: "Fault localization in NoCs by Timed Heartbeats". In Proc. 8th ARCS/VERFE Workshop, GI, LNI 200 GI 2012.

[38] C. Seaton, D. Goodman, M. Lujan, and I. Watson. Applying Dataflow and Transactions to Lee Routing. MULTIPROG, 2012.

[39] C. Christofi, G. Michael, P. Trancoso and P. Evripidou, "Exploring HPC Parallelism with Data-Driven Multithreading". In Proceedings of the International workshop on Dataflow Execution Models for Extreme Scale Computing, 2012.

[40] F. Li, B. Arnoux, and A. Cohen. A compiler and runtime system perspective to scalable dataflow computing. In 5th Works. on Progr. Issues for Heterogeneous Multicores (MULTIPROG), Jan. 2012

[41] S. Weis, A. Garbade, J. Wolf, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer. A Fault Detection and Recovery Architecture for a Teradevice Dataflow System. In: Dataflow Execution Models for Extreme Scale Computing (DFM) Workshop, IEEE, Oct 10, 2011.

[42] A. Garbarde, S. Weis, S. Schlingmann, and T. Ungerer. OC Techniques Applied to Solve Reliability Problems in Future 1000-core Processors. In: Organic Computing — A Paradigm Shift for Complex Systems, pages 575-577. Springer, 2011.

[43] A. Portero, Z. Yu, R. Giorgi, "TERAFLUX: Exploiting Tera-device Computing Challenges", Procedia Computer Science, 7(0), 2011, pp. 146-147.

[44] Z. Yu, A. Righi, R. Giorgi, "A Case Study on the Design Trade-off of a Thread Level Data Flow based Many-core Architecture", Future Computing, Rome, Italy, Sept. 25-30, 2011, pp. 100-106.

[45] R. Giorgi, Z. Popovic, N. Puzovic, "Implementing Fine/Medium Grained TLP Support in a Many-Core Architecture", Proc. 9th Int.l Works. on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2009, Samos, Greece, July 2009, pp. 78-87

[46] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell and J. Planas. OmpSs: A proposal for programming heterogeneous multi-core architectures. In: Parallel Processing Letter, Volume 21, Issue 2, pp. 173 - 193, June 2011.

[47] R. Giorgi, C.A. Prete, G. Prina, L. Ricciardi, "A Hybrid Approach to Trace Generation for Performance Evaluation of Shared-Bus Multiprocessors", IEEE Proc. 22nd EuroMicro Int.l Conf. (EM-96), , Prague, Ceck Republic, Sept. 1996, pp. 207-214.

[48] R. Giorgi, C.A. Prete, G. Prina, L. Ricciardi, "Trace Factory: Generating Workloads for Trace-Driven Simulation of Shared-Bus Multiprocessors", IEEE Concurrency, ISSN:1092-3063, Los Alamitos, CA, USA, vol. 5, no. 4, Oct. 1997, pp. 54-68.

[49] K.M. Kavi, R. Giorgi, J. Arul, "Scheduled dataflow: Execution paradigm, architecture, and performance evaluation" IEEE Trans. Computers, vol. 50, no. 8,:pp. 834–846, Aug. 2001.

[50] A. Pop and A. Cohen. A stream-computing extension to OpenMP. In: Intl. Conf. on High Performance and Embedded Architectures and Compilers (HiPEAC'11), January 24-26, 2011.

[51] F. Li, A. Pop, and A. Cohen. Extending loop distribution to ps-dswp. In: 1st Workshop on Intermediate Representations (WIR'11, associated with CGO), Chamonix, France, April 2011.

[52] K. Trifunović, A. Cohen, R. Ladelski, and F. Li. Elimination of memory-based dependences for loop-nest optimization and parallelization: Evaluation of a revised violated dependence analysis method on a three-address code polyhedral compiler. In: 3rd GCC Research Opportunities Workshop (GROW'11, associated with CGO), Chamonix, France, April 2, 2011.

[53] D. Goodman, B. Khan, S. Khan, C. Kirkham, M. Lujan and I. Watson. MUTS: Native Scala Constructs for Software Transactional Memory. In: Scala Days Workshop, Stanford, June 2-3, 2011.

[54] A. Diavastos, P. Trancoso, M. Luján and I. Watson. Integrating Transactions into the Data-Driven Multi-threading Model using the TFlux Platform. In: Dataflow Execution Models for Extreme Scale Computing (DFM) Workshop, Galveston (Texas), Oct 10, 2011.

[55] S. Schlingmann, A. Garbade, S. Weis, T. Ungerer: "Connectivity-Sensitive Algorithm for Task Placement on a Many-Core Considering Faulty Regions", 2011 19th Euromicro Int.l Conf. on Parallel, Distributed and Network-Based Processing, Cyprus, 9-11 Feb. 2011.

[56] S. Weis, A. Garbade, S. Schlingmann, T. Ungerer: "Towards Fault-Detection Units as an Autonomous Fault Detection Approach for future Many-Cores", 1st Works. Software-controlled, Adaptive Fault-tolerance in Microprocessors (SCAFT), Como, Italy, Feb. 2011.

[57] C. Miranda, A. Pop, P. Dumont, A. Cohen and M. Duranton, "ERBIUM: A Deterministic, Concurrent Intermediate Representation to Map Dataflow Tasks to Scalable, Persistent Streaming Processes". In CASES 2010 Oct. 24-29 2010, Scottsdale, Arizona.

[58] A. Pop, A. Cohen, "Preserving high-level semantics of parallel programming annotations through the compilation flow of optimizing compilers" 15th Workshop on Compilers for Parallel Computers., July 7-9, 2010, Vienna, Austria

[59] Y. Etsion, A. Ramirez, R. M. Badia, E. Ayguade, J. Labarta, and M. Valero, "Task Superscalar: Using Processors as Functional Units", USENIX Workshop on Hot Topics In Parallelism (HotPar), Jun 2010

[60] Y. Etsion, F. Cabarcas, A. Rico, A. Ramirez, R. M. Badia, E. Ayguade, J. Labarta, and M. Valero, "Task Superscalar: An Out-of-Order Task Pipeline". In IEEE Symp. On Microarch. Dec. 2010.

[61] HMPP User's Manual. CAPS enterprise, 2012.

[62] L. Jozwiak, M. Lindwer, R. Corvino, P. Meloni, L. Micconi, J. Madsen, E. Diken, D. Gangadharan, R. Jordans, S. Pomata, P. Pop, G. Tuveri, L. Raffo: ASAM: Automatic Architecture Synthesis and Application Mapping, Proc. DSD 2012 - 15th Euromicro Conf. on Digital System Design, Cesme, , 5-7 Sept. 2012, , pp. 216 - 225.

[63] Y. Jan and L. Jozwiak: Communication and memory architecture design of application-specific high-end multiprocessors, VLSI Design, Vol. 2012, Hindawi Publishing Corporation, doi:10.1155/2012/794753, January 2012, pp. 1 – 20.