Microprocessors and Microsystems 38 (2014) 976–990

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

TERAFLUX: Harnessing dataflow in next generation teradevices

Roberto Giorgi [a,*], Rosa M. Badia [b], François Bodin [c], Albert Cohen [d], Paraskevas Evripidou [e], Paolo Faraboschi [f], Bernhard Fechner [g], Guang R. Gao [h], Arne Garbade [g], Rahul Gayatri [b], Sylvain Girbal [i], Daniel Goodman [j], Behran Khan [j], Souad Koliaï [h], Joshua Landwehr [h], Nhat Minh Lê [d], Feng Li [d], Mikel Lujàn [j], Avi Mendelson [k], Laurent Morin [c], Nacho Navarro [b], Tomasz Patejko [b], Antoniu Pop [d], Pedro Trancoso [e], Theo Ungerer [g], Ian Watson [j], Sebastian Weis [g], Stéphane Zuckerman [h], Mateo Valero [b]

[a] Dip. di Ingegneria dell'Informazione e Scienze Matematiche, Universitá di Siena, Italy
[b] Barcelona Supercomputing Center, Spain
[c] CAPS Enterprise, France
[d] INRIA, France
[e] Dept. of Computer Science, University of Cyprus, Nicosia, Cyprus
[f] Intelligent Infrastructure Lab, Hewlett Packard, Barcelona, Spain
[g] University of Augsburg, Germany
[h] University of Delaware, DE, USA
[i] THALES, France
[j] University of Manchester, United Kingdom
[k] Technion, Israel

ARTICLE INFO

Article history:
Received 4 November 2013
Revised 5 March 2014
Accepted 7 April 2014
Available online 18 April 2014

Keywords:
Dataflow
Programming model
Compilation
Reliability
Architecture
Simulation
Many-cores
Exascale computing
Multi-cores

ABSTRACT

The improvements in semiconductor technologies are gradually enabling extreme-scale systems such as teradevices (i.e., chips composed by 1000 billion of transistors), most likely 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 presents an overview of the research carried out by the TERAFLUX partners and some preliminary results. Our platform comprises 1000+ general purpose cores per chip in order to properly explore the above challenges. An architectural template has been proposed and applications have been ported to the platform. Programming models, compilation tools, and reliability techniques have been developed. The evaluation is carried out by leveraging on modifications of the HP-Labs COTSon simulator.

1. Introduction

Silicon manufacturing technologies, such as FinFET [1] transistors and 3D-die stacking [2] that are currently available, will allow new chips (that we call teradevices) with a huge number of transistors (for current ITRS [3] projections, 1 Tera or 10^{12} transistors), therefore opening the doors to the possibility of exploiting the extremely large amount of parallelism in different ways. It is expected that such systems will be able to perform at least one Exa-FLOPS or 10^{18} floating-point operations per second.

In such 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

* Corresponding author. Tel.: +39 0577 191 5182; fax: +39 0577 195 9064.
E-mail addresses: giorgi@dii.unisi.it (R. Giorgi), rosa.m.badia@bsc.es (R.M. Badia), francois.bodin@caps-entreprise.com (F. Bodin), albert.cohen@inria.fr (A. Cohen), skevos@cs.ucy.ac.cy (P. Evripidou), paolo.faraboschi@hp.com (P. Faraboschi), bernhard.fechner@informatik.uni-augsburg.de (B. Fechner), ggao@capsl.udel.edu (G.R. Gao), arne.garbade@informatik.uni-augsburg.de (A. Garbade), sylvain.girbal@thalesgroup.com (S. Girbal), koliai@eecis.udel.edu (S. Koliaï), josh@eecis.udel.edu (J. Landwehr), mikel.lujan@manchester.ac.uk (M. Lujàn), avi.mendelson@tce.technion.ac.il (A. Mendelson), laurent.morin@caps-entreprise.com (L. Morin), nacho@bsc.es (N. Navarro), antoniu.pop@inria.fr (A. Pop), pedro@cs.ucy.ac.cy (P. Trancoso), theo.ungerer@informatik.uni-augsburg.de (T. Ungerer), watson@cs.man.ac.uk (I. Watson), sebastian.weis@informatik.uni-augsburg.de (S. Weis), szuckerm@eecis.udel.edu (S. Zuckerman), mateo.valero@bsc.es (M. Valero).

http://dx.doi.org/10.1016/j.micpro.2014.04.001
0141-9331/

major challenges: programmability, reliability and complexity of design are here briefly introduced. Also, a new Program eXecution Model [4–6] 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 teradevice) 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 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 [7], 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 [8] 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.

The rest of the paper is organized as follows. Section 2 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 3 describes possible applications based on the OmpSs programming model, while Section 4 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 5) is used for mapping feed-forward dataflow into lower-level dataflow threads as expressed by the T∗ Instruction Set Extension, described in Section 6, together with the architecture of our target system. Section 7 describes the Fault Detection Units (FDUs), which provide fault detection management through monitoring techniques and redundant execution of dataflow threads. The experiments are integrated into a common simulator based on the HPLabs COTSon [9] , presented in Section 8. Finally, Section 9 introduces the codelet model, while Section 10 concludes the paper.

2. General overview of the TERAFLUX project

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

Different layers allow to transform application source code into a dataflow-style binary, and to execute it on the target architecture (which is at the current moment based on off-the-shelf cores like x86_64, even if our approach is Instruction Set agnostic—see Section 8 for more details). The top level of this hierarchy is represented by real world applications, which allow us to stress

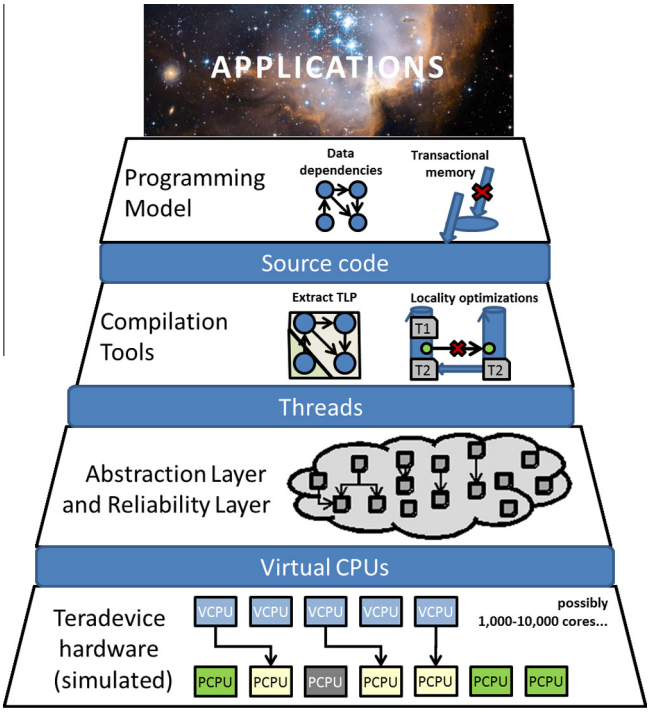


Fig. 1. The TERAFLUX transformation hierarchy.

the underlying teradevice hardware. In the TERAFLUX project, implicit parallelism refers to the set of constraints on the concurrent execution of threads, and the expression of these constraints in the source code. These constraints can be dependencies, 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 [10–12].

A dataflow oriented programming model allows expressing data dependencies among the concurrent tasks of an application. Such concurrent tasks can be subdivided even more—at lower levels—into *DataFlow Threads* (or *DF-Threads*), also simply referred as threads when clear from the context. Nevertheless, applications use large data structures with in-place updates, for efficient memory management [13–15] and copy avoidance. Such applications may 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 [16]. We achieve this by defining a specific layer for studying the integration between the TM and dataflow programming models [17–19].

Besides the programming model, implicit parallelism must be exploited by a compilation tool-chain [20–22], being able to convert dependencies 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 [23,24] programs. The algorithm generalization leverages a new dependence-removal technique to avoid the artificial synchronizations induced by in-place updates in the source program [25,26].

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, such as OpenStream [27]. Both productivity and efficiency programming layers are supported. Compiler directives are used to lower the abstraction penalty of the productivity layer, and to explicitly exploit parallelism and locality.

As mentioned in the Section 1 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 health of the system, and drive specific counteractive measures [28,29]. To achieve this goal in TERAFLUX, we focused on inter-core fault detection techniques using Fault Detection Units (FDUs) [30]. We considered different FDU variants (push, pull, alert mechanisms for heartbeat messages), FDU implementations, and interfaces.

We propose a functional FDU specification based on the MAPE (Monitoring, Analysis, Planning, and Execution) [31] cycle. 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 health 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 hardware structures for scheduling the medium/fine-grain dataflow threads (TSUs or Thread Scheduling Units) 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. For producer-consumer patterns, 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, while non-determinism in accessing the shared data is guaranteed through transactions.

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 system. For the simulation and evaluation of the system we chose a state of the art many-core simulation infrastructure (HPLabs COTSon [9]), 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.

3. Leveraging dataflow through the task-based approach

One of the key aspects of the TERAFLUX project is the proposal of a new programming and execution model [32–34] based on dataflow instead of the traditional control-flow paradigm. Dataflow is known to overcome the limitations of the traditional control-flow model by exploring the maximum parallelism and reducing the synchronization overhead. We leverage such dataflow principle with the combination of OpenStream [27] and StarSs/OmpSs [10–12].

OpenStream compiler — GCC based — is an entry point to TERAFLUX compilation toolchain: applications parallelized with TERAFLUX programming models need to be translated manually or automatically to code annotated with OpenStream directives. We use StarSs memory regions [35] as a case study for translation to OpenStream. OpenStream and StarSs have different features with regard to how data used for computation are represented and how data dependencies are handled:

- OpenStream's basic unit of computation is a *dataflow stream* whereas StarSs applications use *dynamic memory regions* specified by the programmer for communication between tasks
- OpenStream requires explicit task dependencies to maintain correctness of parallel execution whereas data dependencies of StarSs tasks are inferred at runtime.

The comparison shows that translation, when manually done, requires the programmer to identify data dependencies between StarSs tasks and express them with OpenStream streaming constructs. As it is stated in [27], the idea behind StarSs-OpenStream translation scheme is to encode StarSs memory regions as a set of streams that contain versions of memory locations accessed by tasks. The most recent versions in the set of streams are calculated by modified StarSs dependence resolver, and determine live data identified by StarSs memory regions at the given point of application execution. The set of streams is attached to each OpenStream task, and is used by OpenStream runtime to determine data dependencies between tasks and to synchronize concurrent memory accesses. Detailed explanation of the translation scheme and proof of correctness of the algorithm is found in [27].

A source-to-source translator is being developed that carries out StarSs-OpenStream translation at compile time. The key components of the translator are a parser that parses StarSs pragmas to identify memory regions and their directionalities further used to calculate their live versions, and a code generator that generates call expressions to aforementioned StarSs dependence resolver. The code generator also generates OpenStream task pragmas with the set of version streams. Generated code is further passed to OpenStream compiler. A prototype version of this translator is publicly available and has been tested with some sample applications. It can be downloaded from http://openstream.info website.

The StarSs programming model [35] 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 users side there are no explicit parallel constructs, like in thread or stream models.

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 dependencies of the application as well as its potential task parallelism. Recently, an extention of OpenMP was proposed. OmpSs [36,37] is an implementation of StarSs which extends

the OpenMP explicit tasks [38] with dependence clauses that indicate the directionality of the tasks arguments, and on the Task Superscalar design [39,40].

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.

Within 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 website.

3.1. Parallel Updates in StarSs/OmpSs

Although independent tasks from the dependence graph are scheduled for execution, StarSs also provides annotations for simultaneous updates to memory locations shared by multiple tasks. The programmer is responsible to protect such parallel updates. StarSs provides a lock-based synchronization mechanism for concurrency control, and to deal with such concurrent updates. But the use of locks opens the door to deadlock, livelock, nondeterminism, and lost compositionality.

In order to avoid such problems, Software Transactional Memory (STM), an alternative method to lock based synchronization has been used to access shared memory locations. TinySTM [41,42], a lightweight STM library has been integrated into the StarSs framework with this purpose. Instead of introducing a new pragma into the StarSs framework, the implementation of the existing lock pragma was modified to generate transactions which update the shared memory locations. When a lock pragma is encountered StarSs starts a transaction and saves the stack context. If the transaction encounters a conflict at a later stage in the execution then the saved stack context is used to restart the transaction. The critical memory location which is being updated is then loaded into a local variable using TinySTM library calls. The updates are performed on this local copy. At the end the value in this local copy is stored back to the main memory location shared between tasks. In case of a conflict the transaction is restarted from the point where the stack context has been saved. In case of no conflicts the transaction is committed and the results made permanent.

The idea of optimistic STM based synchronization versus pessimistic lock based concurrency control has been tested on applications where parallel updates are performed on memory locations by tasks. The results prove that we obtain higher performance with STM in applications with high lock contention. The overhead of using STM is in the aborts and restarts of transactions in case of a conflict. Hence an analysis has been performed on the time spent by transactions in rollbacks. The results [18] show that in cases where lock based synchronization performs better than STM, the overhead incurred due to rollbacks play a major role. Analysis has been also done on executing longer transactions versus smaller transactions. The trade-off is to create multiple smaller transactions and thus spend more time in start and commit of transactions versus longer transactions and hence longer time in rollbacks in case of a conflict [18].

4. DFScala: constructing and executing dataflow graphs

One part of this project is the construction of a high level dataflow framework which serves two purposes: Overall goals of TERAFLUX included: (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. With these goals in mind, we constructed a high level dataflow framework called DFScala.

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 deterministically based on their input and side-effect

```
#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.

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 dependencies among 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.

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 works [43–45]. DFScala is available at http://apt.cs.man.ac.uk/projects/TERAFLUX/DFScala website.

4.1. Combining dataflow and transactional memory

Transactional memory and dataflow are a good combination of paradigms because of transactions isolation. The detection of conflict and possible retrying is taken care of wholly by the underlying system. As far as the user code is concerned a particular thread sees no evidence of interaction with any other thread, other than a possible increase in execution time. This isolation leads to specific coherency and data dependency properties that fit very well with dataflow programming.

4.2. Coherency

The isolation properties of transactional memory ensure that state updates only become visible at the point that a transaction commits. This means that, unlike with shared state and locks, the coherence model is the same for a transaction as it is for a node of a dataflow computation. As such a transaction can be treated by the memory model as a distinct node of a dataflow graph, so the addition of state does not require a fine grained understanding of the interleaving of operation by the programmer, or strengthening the coherency model in the hardware. A graphical example of this effect can be seen in Fig. 4.

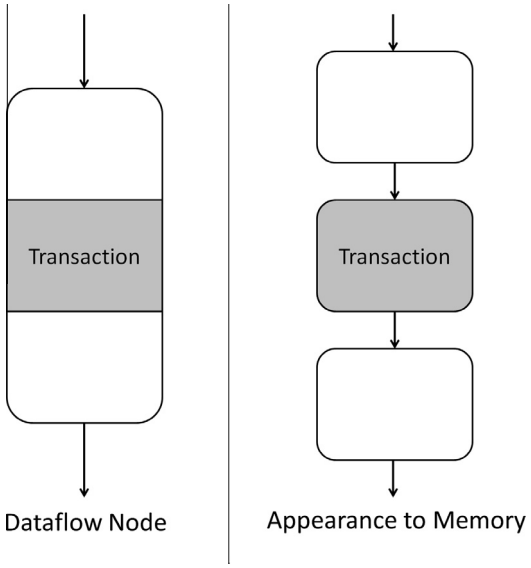


Fig. 4. On the left a node in the dataflow graph containing a transaction, on the right how this can be viewed as three nodes by the memory system.

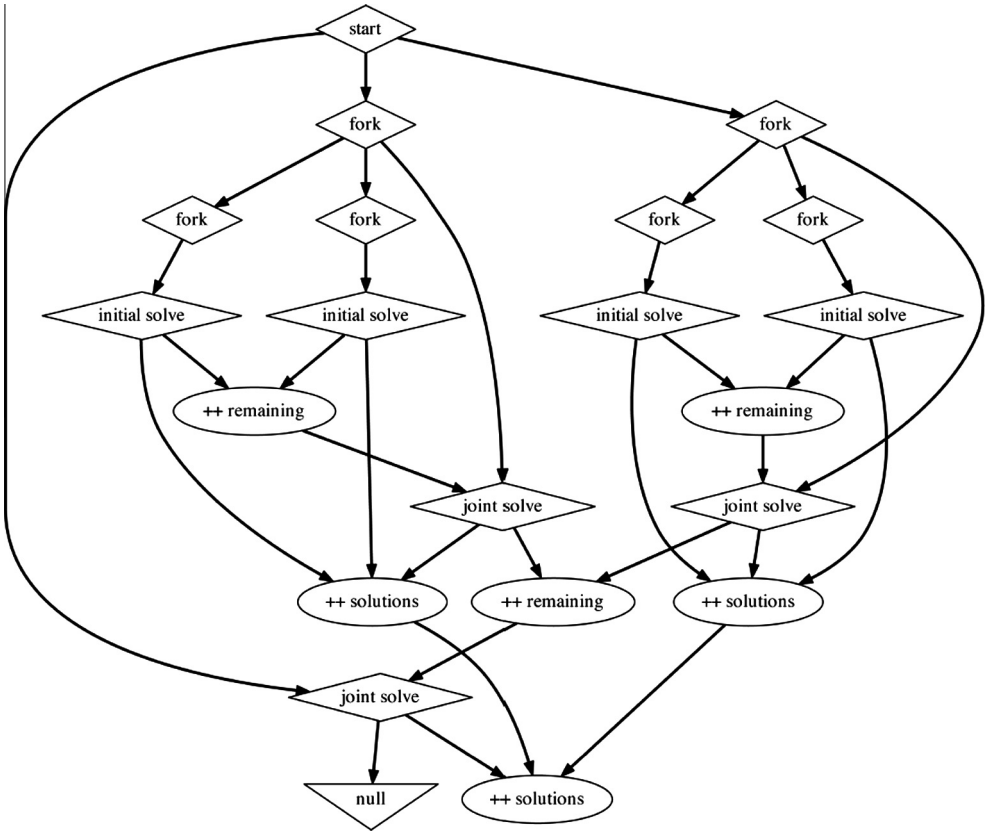


Fig. 3. An instance of a dataflow graph for a circuit routing algorithm.

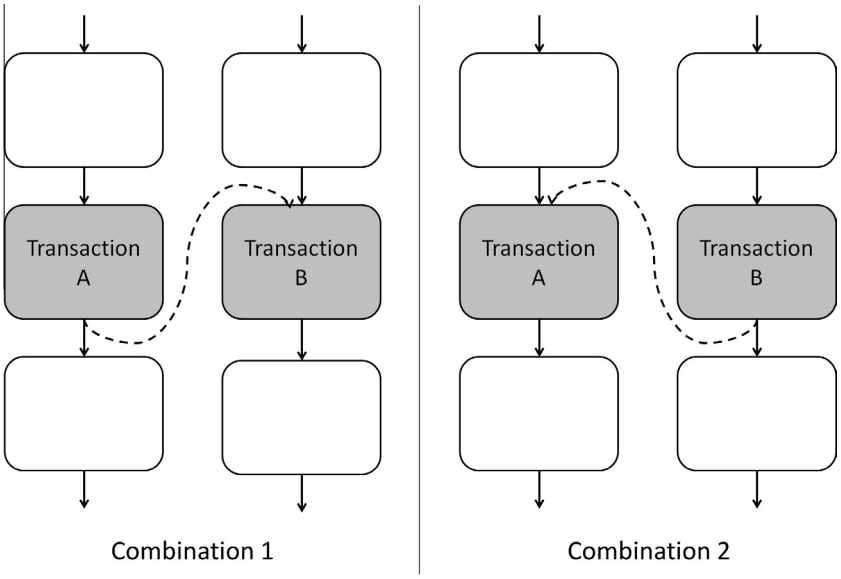


Fig. 5. The two possible formulations of a section of the dataflow graph containing two conflicting transactions. The dashed data dependency will be inserted and enforced at runtime by the transactional memory system, making the relationship between transactions part of the dataflow graph.

4.3. Data dependencies

Isolation also means that transactions can be viewed as dataflow tasks whose data dependencies are determined as the program executes. This is possible because for any two transactions that access the same state and at least one transaction modifies it, they will appear to execute serially. Effectively the TM is adding a dependency to the dataflow graph at runtime. An example of this can be seen in Fig. 5.

The combination of these two features means that transactions can be viewed as nodes in the dataflow graph. This means that we are now describing a family of dataflow graphs which maintain dataflow semantics instead of the weakened dataflow graphs provided by lock based solutions. As such, this combination is deadlock free, like a pure dataflow graph and offers a far smaller number of possible execution paths. This makes it far less invasive to the correctness of the dataflow model while still providing a clean and efficient way to modify state. For example, if we have two transactions each containing 5 lines of code that access shared state, with conventional solutions there would be 252 possible interleavings of the accesses to shared stat $(5 \times 2)!/5!^2$), which would have to be accounted for and locks added to protect them, with transactions there would be 2 possible dataflow graphs. Furthermore if we make an error in our design and use a variable through some back channel, this may cause a performance hit in the form of a bottleneck, but the transaction memory will prevent it causing a race condition.

5. The OpenStream extension to OpenMP

A key point of TERAFLUX is the compilation flow, which 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 [46,27,47], 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 dependencies 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.

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 behavior. 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). The syntax of the additional clauses (a) and an example of stream accessed via sliding window (b) is shown in Fig. 6. OpenStream supports dynamic voltage and frequency scaling under real-time constraints.

We conducted numerous performance evaluations with OpenStream. One key objective of the TERAFLUX project is to confirm the scalability advantages of a dataflow execution model. We study the scalability relative to the number of concurrent tasks created by the program, and relative to the number of cores. We selected the Gauss-Seidel benchmark for its dependence pattern highlighting the benefits of decoupled pipelines for load balancing and scalability. We consider three parallel versions: (1) the manually optimized OpenStream implementation, (2) the systematic conversion of the benchmark to OpenStream using a generic dependence resolver, and (3) the original StarSs benchmark.

The three versions expose the same degree parallelism and their execution unfolds into the same dynamic dependence graph.

The execution time of the three parallel versions as well as the sequential execution, running on 12 cores is shown in Fig. 7. The three parallel versions scale well, and show a slight advantage to dataflow execution with the OpenStream runtime. The reason for the performance difference is not related to the StarSs language or to the implementation of the benchmark. The default StarSs runtime behaves similarly to a token-based dataflow model: it needs to scan for ready tasks. More specifically, the StarSs runtime has a large shared data structure, the tree of regions, which is used to determine whether tasks are ready to execute. When the number of tasks increases, the time spent looking for ready tasks increases, as well as the contention on this data structure. In

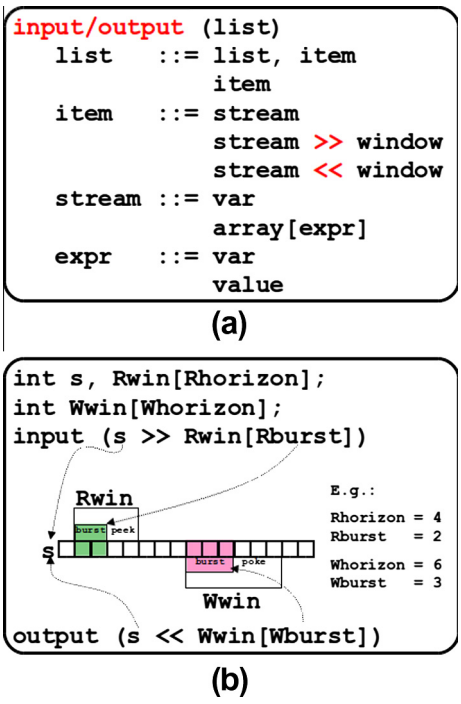


Fig. 6. Syntax for input and output clauses (a) and illustration of stream access through windows (b).

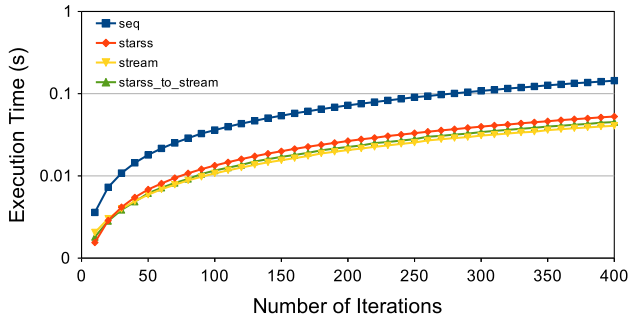


Fig. 7. Gauss-Seidel experiment for a 256 × 256 matrix, with 64 × 64 blocks. The benchmark was run on 12 cores. Four versions executed: OpenStream ('stream'); StarSs ('starss'); an OpenStream-StarSs combination ('starss_to_stream'); and a sequential version ('seq'), used as the baseline.

contrast, the target execution model of OpenStream is feed-forward dataflow, with a scheduler using one ready queue per core and work-stealing. This is possible because the OpenStream compiler generates code spending the extra effort at task creation to find the producer-consumer matching. The producers will then know when consumer tasks become ready, without any polling operation or any lookup in a shared data structure. Once a producer determines that one of its consumers has become ready, it adds it to its core-local, work-stealing ready queue. The systematic translation from the StarSs source code demonstrates that the performance bottlenecks are intrinsic to the runtime implementation and scheduling policy. The task-parallel language design and benchmark implementations can be transparently mapped to a more scalable, dataflow execution model, benefitting from its efficiency and scalability.

6. The TERAFLUX reference architecture

An important aspect of TERAFLUX is represented by the execution model and architecture framework [48] including hardware modules to support the execution model. The proposed template for the TERAFLUX architecture is shown in Fig. 8.

Besides using mostly off-the shelf parts, following the principle of "not reinventing the wheel", the architecture is designed to support an execution model that is a combination of fine- and corse-grain threaded dataflow models including DTA [4], DDM [49], StarSs [32]. In addition, the transactional support has been added to the dataflow model, which allow covering those applications that modify the shared state. Combining dataflow with transactions is a unique feature of this project.

In particular, Data-Driven Multithreading (DDM) [49] 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 (TSU, see Fig. 8) 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.

DDM was applied to three linear algebra HPC applications: Matrix Multiplication, LU decomposition and Cholesky decomposition. The scalability of DDM over a large number of cores was thus tested, and shows encouraging results [50]: DDM can handle the parallelization required for linear algebra applications for present and future multi- and many-core systems, and is thus a viable candidate for HPC.

A distributed version of DDM was also developed [51]. The main difference between single-node and distributed/multi-node DDM execution is the introduction of remote memory accesses resulting from producer and consumer threads running on different nodes. To this end, data forwarding is employed to the node where the consumer is scheduled to run. This is facilitated by supporting a Global Address Space (GAS) across all the nodes. A network interface unit has been implemented in the DDM-TSU to handle the low-level communication operations. In terms of the distribution of threads across the cores of the system nodes, this work explores a static scheme, in which the mapping is determined at compile time and does not change during the execution. The initial results on some kernel like matrix multiplication are very encouraging, leading to an almost linear speedup in configurations up to 32 x86_64 cores.

The TSU design has been also explored in the context of DTA [4]. In this case the TSus are organized in a 2-level hierarchy: one D-TSU at node level and one L-TSU besides each processing unit (standard core). Globally D-TSUs and L-TSUs form what is called the Distributed Thread Scheduler (DTS). In order to support the execution of DF-Threads, a minimalistic extension of the x86_64 ISA was designed, called T-Star (or T-∗) [52].

The key-points of the T-Star Instruction Set Extension (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 the Distributed Thread Scheduler, which monitors the availability of resources and enforces the higher level policies on power and performance constraints; (iii) it enables the usage and management by the DF-Threads of four types of memory regions as requested by the higher levels of the software to support 1-to-1 communication or Thread Local Storage (TLS), N-to-1 communication or Frame Memory (FM), 1-to-N communication or Owner Writable Memory

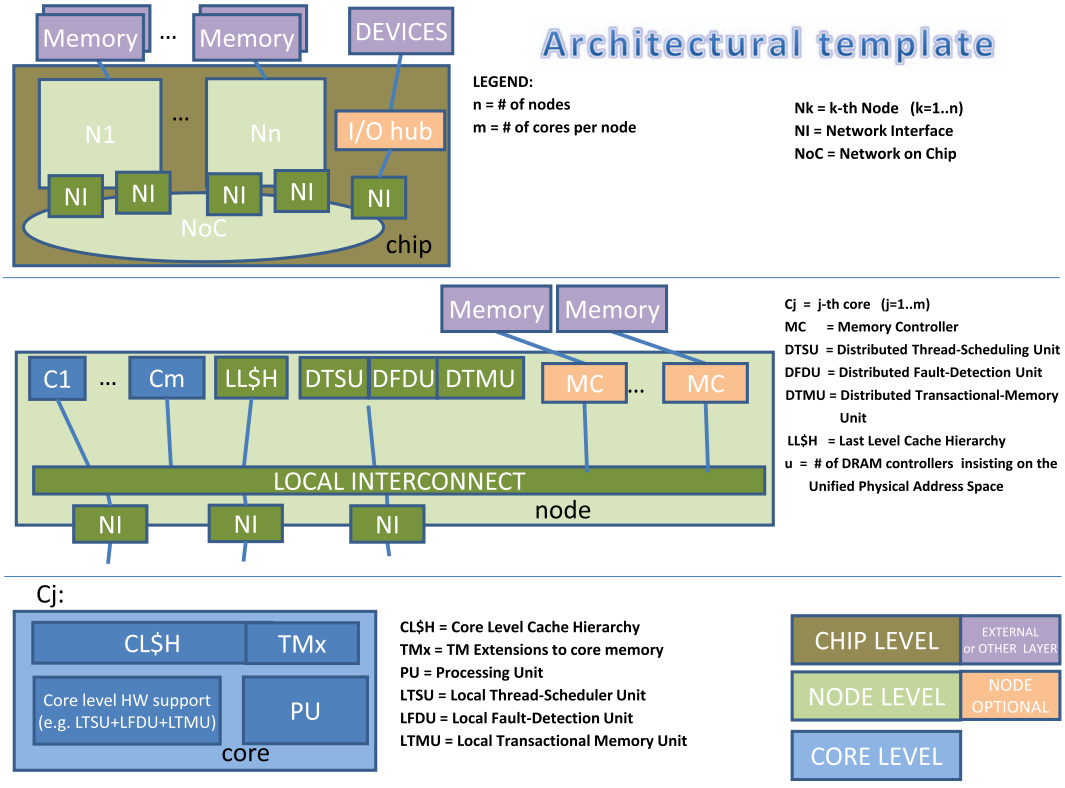


Fig. 8. TERAFLUX architecture template.

(OWM), and N-to-N communication or Transactional Memory (TM).

One feature of DF-Threads supported by the T-Star is their totally dynamic dataflow but without the burden of explicitely managing tokens [53]. The evaluation of this ISE has been carried out in the COTSon simulator: the initial programs were a recursive Fibonacci with input $n = 40$ and a Matrix Multiplication with matrix size 1024×1024 with the initial assumption of executing one instruction per cycle. The results of these test have demonostrated the full scalability of the TERAFLUX architecture up to 1024 cores organized as 32 nodes by 32 cores. The T-Star is available as part of COTSon at http://sf.net/p/cotson/code website.

7. 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. Four levels of reliability aspects are the focus of the TERAFLUX architecture, in order 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). How the D-FDUs relate to a TERAFLUX node is shown in Fig. 9.

In TERAFLUX, the various D-FDUs detect faults by means of the Double Execution mechanism [54,31,55], 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-FDUs 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 may re-execute on different cores. The static dependency graph of a T∗ program (left) and the dynamically created dependency graph of the same program during a Double Execution run (right) is shown in Fig. 10. It can be seen that the original program first executes T0. Since this part of the program is sequential, the TERAFLUX runtime may exploit under-utilized cores for spatial redundant execution of T0. In case of a fault-free execution of T0, the synchronization counts of the successor threads are decremented, and the subsequent threads (T1 T1′), . . . , (Tn Tn′) can be executed.

At the interconnection level, efficient methods have been designed to localize faults within the network (router and link) [56–58]. The localization technique utilizes the knowledge of the

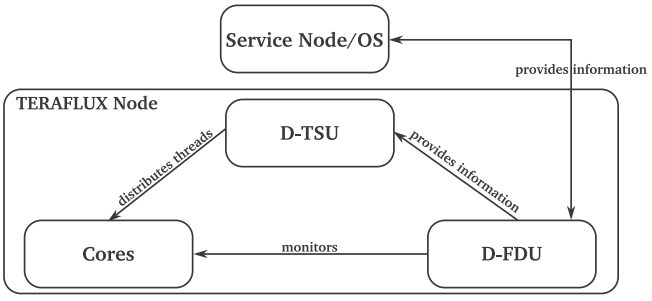


Fig. 9. Interaction between TERAFLUX Nodes, Service Node, Distributed-TSU (D-TSU) and Distributed FDU (D-FDU).

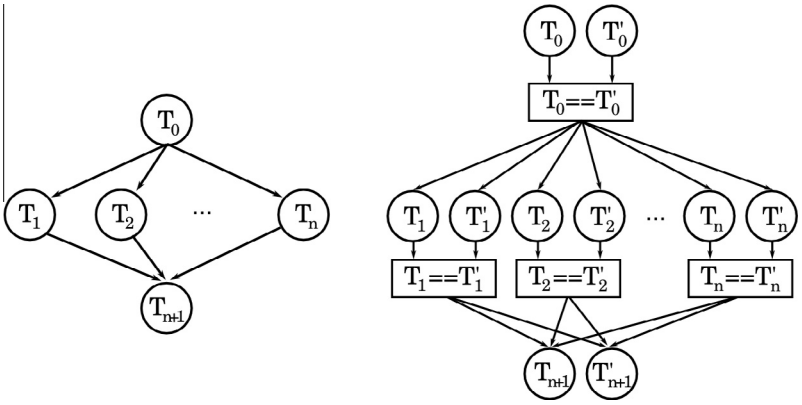


Fig. 10. Dependency graph for regular dataflow execution (left graph) and double execution (right graph).

existing heartbeat messages and extracts inherent information from them. For this, the knowledge of timing information gained from a special message sending pattern is combined with the message path information derived from the routing strategy. The timing pattern was originally designed to isolate heartbeat messages and avoid collisions between them. The pattern regulates the network access for the L-FDUs and is based on the Time Division Multiple Access (TDMA) concept for concurrent media access in computer networks or real time bus systems. In conjunction with the Quality of Service, where heartbeat messages are transferred with the highest priority, it supports a precise estimation of the arrival time of all heartbeat messages at the D-FDU. Supposing that a message was delivered with a delay, indicated by an increased hop count, the receiving D-FDU can conclude that the message was transmitted over a bypass due to a faulty network element.

The routing information is then used to determine where the faulty network component is located. Since the D-FDU investigates an unusual timing behavior of one or more heartbeat messages, the path of these messages is marked as suspicious. Therefore, the D-FDU is holding a network status matrix, which encodes the fault information of the NoC. The suspicious path elements in this matrix are set by incrementing each entry by "1" for each suspicious component. Each heartbeat message arriving in time at the D-FDU ensures that the values corresponding to the message's path are decremented within this matrix. In that manner, a value of smaller than "1" stands for a fault-free component. With this process of elimination, the D-FDU is able to locate faulty network elements.

Finally, the D-FDUs forward the gathered node health states to the operating system to provide additional information for global scheduling decisions.

8. The common evaluation platform

The TERAFLUX project relies on a common evaluation platform [9,59] 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 support tools (e.g., McPAT [60] for power estimation) and libraries (for StarSs region matching) that we integrated to meet our research needs.

An overall picture that highlights what the software sees—as a simulated machine—is shown in Fig. 11. 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) [61,62].

Therefore the simulation software exposes a number of virtual processors where the guest software can run unmodified (called 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 cruncher" (or "Auxiliary Cores"). The simulator may expose the latter capability to the Operating System (OS), but for the sake of generality the application software should not presume the availability of any OS-service: each VCPU is just a bare machine. COTSon, on the other side, can implement any virtualization trick to make this illusion becoming available.

One or more VCPUs assume the role of "Service Cores" (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 run LEGACY APPS (such as the Oracle DBMS). The only support required by the OS is a modification of the internal scheduler (this could be provide as a "driver" in the future) in order to set the high level policies for the TSUs. The TSUs will enforce such policies and take care of the global bookkeping of the various types of threads (DF-Threads, Legacy-Threads (such as from legacy applications),

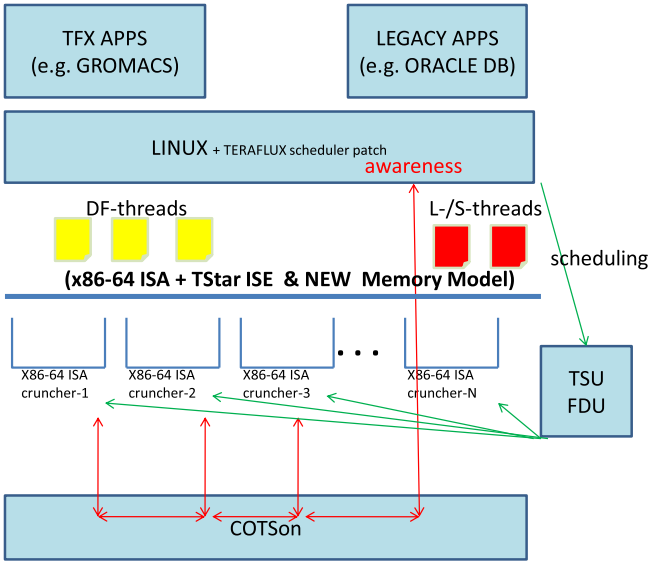


Fig. 11. Overall picture of the "simulator illusion". A number *n* of VCPUs can be used as "workers" or "x86_64 cruncher" or "Auxiliary Cores" or "Service Cores"; a generic *k*th VCPU can be used as service core. L- and S-threads represent Legacy and System Threads, that our system is able to execute.

System-Threads (supporting OS services on the Service Core — respectiely DF-, L-, S-Threads).

We extended the COTSon 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 teradevice system, to a many-nodes-per-chip one, by realizing a communication mechanism via the host shared memory, considering appropriate timing model [63]. More recent steps deal with the further refinement of the T∗ Instruction Set Extension and its integration with the Transactional Memory.

9. The Codelet model

As a further extension of the TERAFLUX program execution model, the University of Delaware joined our project bringing its expertise on the codelet model. The Codelet Model [6] is a hybrid Von Neumann-dataflow execution model, aimed at providing a fine-grain parallel execution model. Its quantum of computation is called a codelet. A codelet (pictured in Fig. 12) is an event-driven sequence of machine instructions, which run until completion. Foremost among those events is data availability, which is the prime reason to trigger the execution of a codelet (which we call firing). All events are explicitly expressed when defining the codelets. Besides the required data (following dataflow semantics), events on which a codelet may wait until it is fired include bandwidth requirements, core frequency (including a given maximal temperature or power expenditure), etc. Codelets are expected to be generated by a compiler. Previous experience with the codelet model's ancestor, EARTH [64], have shown that automatic partitioning programs into threads that follow dataflow semantics is indeed possible [65].

The codelet execution model relies on an Abstract Machine Model (AMM). It features a hierarchical topology, as well as a heterogeneous architecture. From a high-level point of view the codelet AMM is rather close to the TERAFLUX architectural template. The important part is located at the chip level: the scheduling part is delegated to a dedicated unit (the scheduling unit, equivalent of the TSU in the TERAFLUX architectural template), while the computational part is performed by the computation units (which are called cores in TERAFLUX). As a principal scheduling quantum, a codelet, once allocated and scheduled to a core, keeps the computation unit usefully busy, and cannot be preempted (however it can voluntarily yield, provided that more than one codelet context can be held within the same core—thus allowing for overhead-less context switches). One feature of the codelet execution is the efficient support of a non-preemptive event-driven thread model. If the system software deems it necessary, it can clock-gate (or even power-gate) selected cores. The codelets running on those cores must thus be suspended for the duration of the core suspension—and subsequently resumed once the TSU it depends on decides to turn the core on again. Codelets running on cores that are power-gated must be restarted (either on the same core or on another).

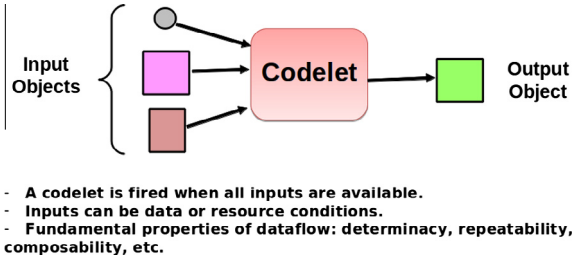


Fig. 12. The Codelet: a Fine-Grain Piece of Computation. A codelet is fired when all inputs are available. Inputs can be data (square inputs) or resource conditions (gray circle).

The codelet model features asynchronous functions, called Threaded Procedures. They are called in a control-flow manner, and act as containers for codelet graphs. A threaded procedure (TP) features a frame which is shared by all the codelets. Following the precepts of dataflow, when possible, most data is written in a "write once, read many times" manner: data is produced by a given codelet, and consumed by one or more depending codelets. Threaded procedures provide a second level of parallelism, as well as a locality constraint: all codelets contained within a given TP must run on the same group of cores within a node. Therefore, while there is nothing to prevent the grouping of unrelated codelets within a given TP, locality requirements tend to ensure that TPs are containers for a logical grouping of codelets, much like traditional functions are logical groupings of instructions. The availability of the TP frame to all its contained codelets implies that all codelets can read and write all its data: two codelets executing in parallel may potentially read and/or write to the same frame location simultaneously. Therefore a memory consistency model must be provided to handle those cases.

The codelet memory model is based on Location Consistency (LC) [66]. 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.

We have provided an implementation of the Codelet model as a runtime system: the Delaware Adaptive Run-Time System (DARTS) [67]. It is written in C++, and can run on any POSIX-compliant system. In addition, we have ported DARTS to the TERAFLUX simulator, thus taking advantage of the TERAFLUX instruction set extension to schedule dataflow threads. The codelet and DF-Thread models are sufficiently close that we can try to find a converging path toward a new dataflow-inspired execution model, of which the port of DARTS on the TERAFLUX machine (the COTSon simulator) is a first significant step. Codelets are mapped to DF-Threads, thus allowing DARTS to make use of the TSU on COTSon. We are also studying different percolation techniques for teradevices. Percolation [68] is a mechanism that determines how code and/or data should be located, and where, on a given machine. It also takes into account the bandwidth availability. Its goal is to guarantee as much locality as possible for the on-going computation. Other research is currently under way to combine the Codelet model with streaming. Preliminary results are encouraging [69,70].

Other projects seem to move along similar directions [71,72].

10. Conclusions

We presented TERAFLUX, a Future Emerging Technology Large-Scale Project funded by the EU. TERAFLUX is at the forefront of major research challenges such as programmability, manageable architecture design, and reliability of many-core or 1000+ cores chips. We described the project's transformation hierarchy and its major scientific contributions. These contributions include scalable parallel applications, programming models for large-scale systems exploiting the dataflow principles, compiler techniques and tools to harness a dynamic, thread-level dataflow execution model, fault-detection and -tolerance techniques, the actual architecture, and the simulation infrastructure needed for the evaluation of

the proposed research. The preliminary results demonstrate solid performance scalability and reliable execution over unreliable components. The TERAFLUX architecture builds on top of classical microarchitecture principles for the individual cores and their closest local memories, and combines a large number of them with simple modules such as the Thread Scheduling Units and Fault Detection Units. Among the notable contributions, StarSs-inpired dependent tasking construct have been integrated into the OpenMP4 specification, and the project released a simulator able to perform experiments with more the 1000 general purpose cores and full system simulation (COTSon).

Acknowledgements

This work was partly funded by the European FP7 Project TERAFLUX (id. 249013) http://www.teraflux.eu. Prof. Avi Mendelsons work has been carried out at Microsoft R&D, Israel. The researchers from BSC also acknowledge the support of the Spanish Ministry of Science and Innovation (Contract TIN2012-34557) and of the Generalitat de Catalunya (Contract 2009-SGR-980).

References

[1] M. Jurczak, N. Collaert, A. Veloso, T. Hoffmann, S. Biesemans, Review of FINFET technology, in: SOI Conference, 2009 IEEE International, 2009, pp. 1–4. <http://dx.doi.org/10.1109/SOI.2009.5318794>.

[2] R. Chanchani, 3D integration technologies – an overview, in: D. Lu, C. Wong, (Eds.), Materials for Advanced Packaging, Springer US, 2009, pp. 1–50. doi: 10.1007/978-0-387-78219-5_1. <http://dx.doi.org/10.1007/978-0-387-78219-5_1>.

[3] International Technology Roadmap for Semiconductors. <http://www.itrs.net/Links/2011ITRS/Home2011.htm>.

[4] R. Giorgi, Z. Popovic, N. Puzovic, DTA-C: a decoupled multi-threaded architecture for CMP systems, in: 19th International Symposium on Computer Architecture and High Performance Computing, 2007, SBAC-PAD 2007, 2007, pp. 263–270. <http://dx.doi.org/10.1109/SBAC-PAD.2007.27>.

[5] K. Kavi, R. Giorgi, J. Arul, Scheduled dataflow: execution paradigm, architecture, and performance evaluation, IEEE Trans. Comput. 50 (8) (2001) 834–846, http://dx.doi.org/10.1109/12.947003.

[6] S. Zuckerman, J. Suetterlein, R. Knauerhase, 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, ACM, New York, NY, USA, 2011, pp. 64–69. <http://dx.doi.org/10.1145/2000417.2000424. URL http://doi.acm.org/10.1145/2000417.2000424>.

[7] J.B. Dennis, The Data Flow Concept Past, Present and Future, Keynote Speech at the Dataflow Execution Models for Extreme Exascale Computing Workshop, October 2011.

[8] A. Portero, Z. Yu, R. Giorgi, Teraflux: exploiting tera-device computing challenges, Procedia Comput. Sci. 7 (0) (2011) 146–147 (Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11). doi: http://dx.doi.org/10.1016/j.procs.2011.09.081).

[9] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, D. Ortega, COTSon: infrastructure for full system simulation, SIGOPS Oper. Syst. Rev. 43 (1) (2009) 52–61, http://dx.doi.org/10.1145/1496909.1496921. <http://doi.acm.org/10.1145/1496909.1496921>.

[10] J. Ciesko, J. Bueno, N. Puzovic, A. Ramirez, R. Badia, J. Labarta, Programmable and scalable reductions on clusters, in: 2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), 2013, pp. 560–568, doi: 10.1109/IPDPS.2013.63.

[11] J. Planas, R. Badia, E. Ayguade, J. Labarta, Self-adaptive ompss tasks in heterogeneous environments, in: 2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), 2013, pp. 138–149, doi: 10.1109/IPDPS.2013.53.

[12] F. Yazdanpanah, D. Jimenez-Gonzalez, C. Alvarez-Martinez, Y. Etsion, R.M. Badia, Fpga-based prototype of the task superscalar architecture, in: 7th HiPEAC Workshop on Reconfigurable Computing (WRC 2013), Berlin, Germany, 2013.

[13] N.M. Lê, A. Pop, A. Cohen, F. Zappa Nardelli, Correct and efficient work-stealing for weak memory models, SIGPLAN Not. 48 (8) (2013) 69–80, http://dx.doi.org/10.1145/2517327.2442524. <http://doi.acm.org/10.1145/2517327.2442524>.

[14] B. Diouf, C. Hantaş, A. Cohen, O. Özturk, J. Palsberg, A decoupled local memory allocator, ACM Trans. Architec. Code Optim. 9 (4) (2013) 34:1–34:22, http://dx.doi.org/10.1145/2400682.2400693. <http://doi.acm.org/10.1145/2400682.2400693>.

[15] K. Trifunovic, A. Cohen, R. Ladelsky, 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: Proceedings of 3rd International Workshop on GCC Research Opportunities (GROW 2011), 2011. <http://grow2011.inria.fr/>.

[16] Seaton, Chris and Goodman, Daniel and Luján, Mikel and Watson, Ian, Applying Dataflow and Transactions to Lee Routing, in: Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG), Paris, France, 2012.

[17] I. Herath, D. Rosas-Ham, D. Goodman, M. Luján, I. Watson, A Case for Exiting a Transaction in the Context of Hardware Transactional Memory.

[18] Rahulkumar Gayatri, Rosa M. Badia, Eduard Ayguade, Mikel Luján, Ian Watson, Transactional access to shared memory in StarSs, a task based programming model, in: C. Kaklamanis, T. Papatheodorou, P.G. Spirakis (Eds.), Euro-Par 2012 Parallel Processing, Lecture Notes in Computer Science, vol. 7484, Springer, Berlin Heidelberg, 2012, pp. 514–525. doi: 10.1007/978-3-642-32820-6_51, <http://dx.doi.org/10.1007/978-3-642-32820-6_51>.

[19] A. Diavastos, P. Trancoso, M. Lujan, I. Watson, Integrating transactions into the data-driven multi-threading model using the TFlux platform, in: 2011 First Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM), 2011, pp. 19–27, doi: 10.1109/DFM.2011.14.

[20] F. Li, B. Arnoux, A. Cohen, A compiler and runtime system perspective to scalable data-flow computing, in: Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG), Paris, France, 2012.

[21] C. Miranda, A. Pop, P. Dumont, A. Cohen, M. Duranton, Erbium: a deterministic, concurrent intermediate representation to map data-flow tasks to scalable, persistent streaming processes, in: Proceedings of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES '10, ACM, New York, NY, USA, 2010, pp. 11–20. doi: 10.1145/1878921.1878924, <http://doi.acm.org/10.1145/1878921.1878924>.

[22] A. Pop, A. Cohen, Preserving high-level semantics of parallel programming annotations through the compilation flow of optimizing compilers, in: Proceedings of the 15th Workshop on Compilers for Parallel Computers (CPC'10), Vienna, Autriche, 2010. <http://hal.inria.fr/inria-00551518>.

[23] CAPS Entreprise, HMPP Directives Reference Manual, Version 3.2.0, CAPS entreprise, 2012.

[24] OpenHMPP Consortium Association, OpenHMPP New Standard for Many-Core, OpenHMPP Consortium Association, 2011.

[25] F. Li, A. Pop, A. Cohen, Automatic extraction of coarse-grained data-flow threads from imperative programs, IEEE Micro 32 (4) (2012) 19–31, http://dx.doi.org/10.1109/MM.2012.49.

[26] F. Li, A. Pop, A. Cohen, Extending loop distribution to PS-DSWP, in: F. Bouchez, S. Hack, E. Visser (Eds.), Proceedings of the Workshop on Intermediate Representations, 2011, pp. 29–36. <http://researchr.org/publication/Li-WIR-2011>.

[27] A. Pop, A. Cohen, OpenStream: expressiveness and data-flow compilation of OpenMP streaming programs, ACM Trans. Architec. Code Optim. 9 (4) (2013) 53:1–53:25, http://dx.doi.org/10.1145/2400682.2400712. <http://doi.acm.org/10.1145/2400682.2400712>.

[28] J. Wolf, B. Fechner, T. Ungerer, Fault coverage of a timing and control flow checker for hard real-time systems, in: On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International, 2012, pp. 127–129, doi: 10.1109/IOLTS.2012.6313855.

[29] J. Wolf, B. Fechner, S. Uhrig, T. Ungerer, Fine-grained timing and control flow error checking for hard real-time task execution, in: 2012 7th IEEE International Symposium on Industrial Embedded Systems (SIES), 2012, pp. 257–266, doi: 10.1109/SIES.2012.6356592.

[30] S. Weis, A. Garbade, S. Schlingmann, T. Ungerer, Towards Fault Detection Units as an Autonomous Fault Detection Approach for Future Many-Cores, in: ARCS Workshops, 2011.

[31] A. Garbade, S. Weis, S. Schlingmann, T. Ungerer, OC techniques applied to solve reliability problems in future 1000-core processors, in: C. Müller-Schloer, H. Schmeck, T. Ungerer (Eds.), Organic Computing — A Paradigm Shift for Complex Systems, Autonomic Systems, vol. 1, Springer, Basel, 2011, pp. 575–577. doi: 10.1007/978-3-0348-0130-0_38. <http://dx.doi.org/10.1007/978-3-0348-0130-0_38>.

[32] J. Bueno, J. Planas, A. Duran, R. Badia, X. Martorell, E. Ayguade, J. Labarta, Productive programming of GPU clusters with OmpSs, in: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, 2012, pp. 557–568, doi: 10.1109/IPDPS.2012.58.

[33] V.K. Elangovan, R. Badia, E.A. Parra, OmpSs-OpenCL programming model for heterogeneous systems, in: H. Kasahara, K. Kimura (Eds.), Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science, vol. 7760, Springer, Berlin Heidelberg, 2013, pp. 96–111. doi: 10.1007/978-3-642-37658-0_7. <http://dx.doi.org/10.1007/978-3-642-37658-0_7>.

[34] R. Giorgi, Z. Popovic, N. Puzovic, Implementing fine/medium grained TLP support in a many-core architecture, in: K. Bertels, N. Dimopoulos, C. Silvano, S. Wong (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation, Lecture Notes in Computer Science, vol. 5657, Springer, Berlin Heidelberg, 2009, pp. 78–87. doi: 10.1007/978-3-642-03138-0_9. <http://dx.doi.org/10.1007/978-3-642-03138-0_9>.

[35] J. Planas, R.M. Badia, E. Ayguadé, J. Labarta, Hierarchical task-based programming with StarSs, Int. J. High Perform. Comput. Appl. 23 (3) (2009) 284–299. http://arxiv.org/abs/http://hpc.sagepub.com/content/23/3/284.full.pdf+html arXiv:http://hpc.sagepub.com/content/23/3/284.full.pdf+html, http://dx.doi.org/10.1177/1094342009106195, doi: 10.1177/1094342009106195. <http://hpc.sagepub.com/content/23/3/284.abstract>.

[36] A. Duran, R. Ferrer, E. Ayguadé, R.M. Badia, J. Labarta, A proposal to extend the openmp tasking model with dependent tasks, Int. J. Parallel Prog. 37 (3) (2009)

292–305, http://dx.doi.org/10.1007/s10766-009-0101-1. <http://dx.doi.org/10.1007/s10766-009-0101-1>.
[37] A. Duran, E. Ayguadé, R.M. Badia, J. Labarta, L. Martinell, X. Martorell, J. Planas, OmpSs: a proposal for programming heterogeneous multi-core architectures, Parallel Process. Lett. 21 (02) (2011) 173–193, http://dx.doi.org/10.1142/S0129626411000151. http://arxiv.org/abs/http://www.worldscientific.com/doi/pdf/10.1142/S0129626411000151 arXiv:http://www.worldscientific.com/doi/pdf/10.1142/S0129626411000151, doi: 10.1142/S0129626411000151. <http://www.worldscientific.com/doi/abs/10.1142/S0129626411000151>.
[38] OpenMP Architecture Review Board, OpenMP Application Program Interface, Specification (2011). <http://www.openmp.org/mp-documents/OpenMP3.1.pdf>.
[39] Y. Etsion, A. Ramirez, R.M. Badia, E. Ayguade, J. Labarta, M. Valero, Task superscalar: using processors as functional units, in: USENIX Workshop on Hot Topics In Parallelism (HotPar). Berkeley, CA, USA, vol. 35, 2010.
[40] Y. Etsion, F. Cabarcas, A. Rico, A. Ramirez, R. Badia, E. Ayguade, J. Labarta, M. Valero, Task superscalar: an out-of-order task pipeline, in: Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on, 2010, pp. 89–100, doi: 10.1109/MICRO.2010.13.
[41] P. Felber, C. Fetzer, P. Marlier, T. Riegel, Time-based software transactional memory, IEEE Trans. Parallel Distrib. Syst. 21 (12) (2010) 1793–1807, http://dx.doi.org/10.1109/TPDS.2010.49.
[42] P. Felber, C. Fetzer, T. Riegel, Dynamic performance tuning of word-based software transactional memory, in: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '08, ACM, New York, NY, USA, 2008, pp. 237–246. doi: 10.1145/1345206.1345241. <http://doi.acm.org/10.1145/1345206.1345241>.
[43] D. Goodman, B. Khan, S. Khan, M. Luján, I. Watson, Software transactional memories for scala, J. Parallel Distrib. Comput. 73 (2) (2013) 150–163. http://dx.doi.org/http://dx.doi.org/10.1016/j.jpdc.2012.09.015 doi: http://dx.doi.org/10.1016/j.jpdc.2012.09.015. <http://www.sciencedirect.com/science/article/pii/S0743731512002304>.
[44] D. Goodman, S. Khan, C. Seaton, Y. Guskov, B. Khan, M. Luján, I. Watson, DFScala: high level dataflow support for scala, in: Second International Workshop on Data-Flow Models For Extreme Scale Computing (DFM), Minneapolis, MN, USA, 2012.
[45] Goodman Daniel, Khan Behram, Khan Salman, Kirkham Chris, Luján Mikel, Watson Ian, MUTS: native scala constructs for software transactional memory, in: Proceedings of Scala Days 2011, Stanford CA, 2011.
[46] A. Pop, A. Cohen, Work-streaming compilation of futures, in: 5th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), Tallin, Estonia, 2012.
[47] A. Pop, A. Cohen, A stream-computing extension to OpenMP, in: Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers, HiPEAC '11, ACM, New York, NY, USA, 2011, pp. 5–14. doi: 10.1145/1944862.1944867. <http://doi.acm.org/10.1145/1944862.1944867>.
[48] Z. Yu, A. Righi, R. Giorgi, A case study on the design trade-off of a thread level data flow based many-core architecture, in: FUTURE COMPUTING 2011, The Third International Conference on Future Computational Technologies and Applications, 2011, pp. 100–106.
[49] C. Kyriacou, P. Evripidou, P. Trancoso, Data-driven multithreading using conventional microprocessors, IEEE Trans. Parallel Distrib. Syst. 17 (10) (2006) 1176–1188, http://dx.doi.org/10.1109/TPDS.2006.136.
[50] C. Christofi, G. Michael, P. Trancoso, P. Evripidou, Exploring HPC parallelism with data-driven multithreating, in: Data-Flow Execution Models for Extreme Scale Computing (DFM), 2012, 2012, pp. 10–17, doi: 10.1109/DFM.2012.11.
[51] G. Michael, S. Arandi, P. Evripidou, Data-flow concurrency on distributed multi-core systems, in: In Proceedings of the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '13), July 22–25, 2013, Las Vegas, USA, 2013.
[52] R. Giorgi, TERAFLUX: exploiting dataflow parallelism in teradevices, in: Proceedings of the 9th Conference on Computing Frontiers, CF '12, ACM, New York, NY, USA, 2012, pp. 303–304. doi: 10.1145/2212908.2212959. <http://doi.acm.org/10.1145/2212908.2212959>.
[53] Arvind, R.S. Nikhil, Executing a program on the mit tagged-token dataflow architecture, IEEE Trans. Comput. 39 (3) (1990) 300–318, http://dx.doi.org/10.1109/12.48862.
[54] S. Weis, A. Garbade, J. Wolf, B. Fechner, A. Mendelson, R. Giorgi, T. Ungerer, A fault detection and recovery architecture for a teradevice dataflow system, in: 2011 First Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM), 2011, pp. 38–44, doi: 10.1109/DFM.2011.9.
[55] S. Weis, A. Garbade, S. Schlingmann, T. Ungerer, Towards fault detection units as an autonomous fault detection approach for future many-cores, in: ARCS Workshops, 2011.
[56] A. Garbade, S. Weis, S. Schlingmann, B. Fechner, T. Ungerer, Impact of message based fault detectors on applications messages in a network on chip, in: 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2013, pp. 470–477, doi: 10.1109/PDP.2013.76.
[57] A. Garbade, S. Weis, S. Schlingmann, B. Fechner, T. Ungerer, Fault Localization in NoCs Exploiting Periodic Heartbeat Messages in a Many-Core Environment, in: Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, 2013, pp. 791–795, doi: 10.1109/IPDPSW.2013.150.
[58] S. Schlingmann, A. Garbade, S. Weis, T. Ungerer, Connectivity-sensitive algorithm for task placement on a many-core considering faulty regions, in: PDP, 2011, pp. 417–422, doi: 10.1109/PDP.2011.58.
[59] 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, in: Proceedings of the 45th Annual Simulation Symposium, ANSS '12, Society for Computer Simulation International, San Diego, CA, USA, 2012, pp. 9:1–9:7. <http://dl.acm.org/citation.cfm?id=2331751.2331760>.
[60] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, N. Jouppi, Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures, in: 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009, MICRO-42, 2009, pp. 469–480.
[61] R. Giorgi, C. Prete, L. Ricciardi, G. Prina, A hybrid approach to trace generation for performance evaluation of shared-bus multiprocessors, in: EUROMICRO 96. Beyond 2000: Hardware and Software Design Strategies., Proceedings of the 22nd EUROMICRO Conference, 1996, pp. 207–214, doi: 10.1109/EURMIC.1996.546384.
[62] R. Giorgi, C. Prete, G. Prina, L. Ricciardi, Trace factory: generating workloads for trace-driven simulation of shared-bus multiprocessors, IEEE Concurrency 5 (4) (1997) 54–68, http://dx.doi.org/10.1109/4434.641627.
[63] J. Navaridas, B. Khan, S. Khan, P. Faraboschi, M. Lujan, Reservation-based network-on-chip timing models for large-scale architectural simulation, in: Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on, 2012, pp. 91–98, doi: 10.1109/NOCS.2012.18.
[64] H.H.J. Hum, O. Maquelin, K.B. Theobald, X. Tian, G.R. Gao, L.J. Hendren, A study of the EARTH-MANNA multithreaded system, Int. J. Parallel Program. 24 (4) (1996) 319–348. <http://dl.acm.org/citation.cfm?id=239178.239180>.
[65] L. Hendren, X. Tang, Y. Zhu, G. Gao, X. Xue, H. Cai, P. Ouellet, Compiling C for the EARTH multithreaded architecture, in: Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques, 1996, 1996, pp. 12–23, doi: 10.1109/PACT.1996.552551.
[66] G. Gao, V. Sarkar, Location consistency-a new memory model and cache consistency protocol, IEEE Trans. Comput. 49 (8) (2000) 798–813, http://dx.doi.org/10.1109/12.868026.
[67] J. Suettlerlein, S. Zuckerman, G. Gao, An implementation of the codelet model, in: F. Wolf, B. Mohr, D. Mey (Eds.), Euro-Par 2013 Parallel Processing, Lecture Notes in Computer Science, vol. 8097, Springer, Berlin Heidelberg, 2013, pp. 633–644. doi: 10.1007/978-3-642-40047-6_63. <http://dx.doi.org/10.1007/978-3-642-40047-6_63>.
[68] G. Gao, K. Likharev, P. Messina, T. Sterling, Hybrid technology multithreaded architecture, in: Frontiers of Massively Parallel Computing, 1996, Proceedings Frontiers '96, Sixth Symposium on the, 1996, pp. 98–105, doi: 10.1109/FMPC.1996.558066.
[69] H. Wei, J. Yu, H. Yu, M. Qin, G.R. Gao, Software pipelining for stream programs on resource constrained multicore architectures, IEEE Trans. Parallel Distrib. Syst. 23 (12) (2012) 2338–2350. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.41 doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.41.
[70] H. Wei, G.R. Gao, W. Zhang, J. Yu, COStream: a dataflow programming language and compiler for multi-core architecture, in: To appear in Proceedings of Data-Flow Models (DFM) for Extreme Scale Computing Workshop 2013 in conjunction with Parallel Architectures and Compilation Technologies (PACT 2013), Edinburgh, Scotland, UK, 2013.
[71] Y. Jan, L. Jóźwiak, Communication and memory architecture design of application-specific high-end multiprocessors, VLSI Des. 20 (2012), http://dx.doi.org/10.1155/2012/794753 (12:12–12:12). <http://dx.doi.org/10.1155/2012/794753>.
[72] 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, in: 2012 15th Euromicro Conference on Digital System Design (DSD), 2012, pp. 216–225, doi: 10.1109/DSD.2012.28.

Roberto Giorgi is an Associate Professor at Department of Information Engineering, University of Siena, Italy. He received his PhD in Computer Engineering and his Master in Electronics Engineering, Magna cum Laude both from University of Pisa. He is the coordinator of the European Project TERAFLUX, Future and Emerging Technologies for Teradevice Computing. He is a member of the Network of Excellence HiPEAC (High Performance Embedded-system Architecture and Compiler). He contributed to ERA (Embedded Reconfigurable Architectures), SARC (Scalable ARChitectures), ChARM (caChe for ARM based systems) projects. His interests include Computer Architecture themes such as Embedded Systems, Multiprocessors, Memory System, Workload Characterization.

Rosa M. Badia holds a PhD on Computer Science (1994) from the Technical University of Catalonia (UPC). She is a Scientific Researcher from the Consejo Superior de Investigaciones Científicas (CSIC) and team leader of the Grid Computing and Cluster research group at the Barcelona Supercomputing Center (BSC). She was involved in teaching and research activities at the UPC from 1989 to 2008, as Associated Professor since year 1997. From 1999 to 2005 she was involved in research and development activities at the European Center of Parallelism of Barcelona (CEPBA). Her current research interest are programming models for complex platforms.

François Bodin held various research positions at University of Rennes I and at the INRIA research lab. His contribution includes new approaches for exploiting high performance processors in scientific computing and in embedded applications. He cofounded CAPS (www.caps-entreprise.com) in 2002 while he was a Professor at University of Rennes I and in January 2008 he becomes the company CTO. He is also at the origin of another company, TOCEA (http://tocea.com) that focuses on control of source code quality and refactoring. In 2013 he is back at Irisa (http://www.irisa.fr) on a Professor position.

Albert Cohen is a senior research scientist at INRIA and a part-time associate professor at École Polytechnique. He graduated from École Normale Supérieure de Lyon, and received his PhD from the University of Versailles in 1999 (awarded two national prizes). He has been a visiting scholar at the University of Illinois and an invited professor at Philips Research. Albert Cohen works on parallelizing and optimizing compilers, parallel programming, and synchronous programming for embedded systems. Several research projects initiated or led by Albert Cohen resulted in the transfer of advanced compilation techniques to production compilers.

Paraskevas Evripidou is Professor of Computer Science at the University of Cyprus. He received the HND in Electrical Engineering from the Higher Technical Institute in Nicosia Cyprus in 1981. He received the BSEE, MS and PhD Computer Engineering in 1985, 1986 and 1990 respectively from the University of Southern California. His research interests are: Parallel Processing, Computer Architecture with emphasis on Data-Flow systems, and also works on Ubiquitous/Pervasive computing. He is co-founder and the Chair of the Cyprus Entrepreneurship Competition (CyEC) http://www.cyec.org.cy. He is a member of the IFIP Working Group 10.3, the IEEE Computer Society and ACM SIGARCH.

Paolo Faraboschi is a Distinguished Technologist at HP Labs, working on low-energy servers and project Moonshot. His research interests are at the intersection of architecture and software. From 2004 to 2009, he led the HPL research activity on system-level simulation. From 1995 to 2003, he was the principal architect of the Lx/ST200 family of VLIW cores, widely used in video SoCs and HP's printers. Paolo is an IEEE Fellow and an active member of the computer architecture community. He holds 24 patents and a PhD in EECS from the University of Genoa, Italy.

Bernhard Fechner holds a PhD in computer science from the University of Hagen and is IEEE senior member. He has (co-)authored over 40 refereed papers in the field of fault-tolerant computing in conference proceedings, workshops and journals. Bernhard is speaker of the Fault Tolerant Computing Section of the German Computer Science Society (GI). Currently he is working as assistant professor at the University of Augsburg.

Guang R. Gao is Endowed Distinguished Professor in the Department of Electrical and Computer Engineering and head of the Computer Architecture and Parallel Systems Laboratory at the University of Delaware. He is an ACM and IEEE fellow. He has been advocating the use of dataflow principles for the past twenty years. He is one of the Principal Investigators in the DOE-funded eXtreme-scale software Stack (X-Stack) project. His research areas include computer architecture and parallel systems, optimizing and parallelizing compilers, runtime systems, and their application to bio-informatics and high-performance computing.

Arne Garbade received the Diploma degree in computer science from the University of Bremen, Germany, in 2009. He is currently a PhD student an the University of Augsburg. His topics of interest include fault tolerance techniques, adaptive systems, and on-chip communication networks in the many-core domain.

Rahulkumar Gayatri is a doctoral student at the Barcelona Supercomputing Center. He finished his masters in India at the Sri Sathya Sai University in March 2009. He joined the doctoral program in September 2009. He is a part of the Teraflux project.

Sylvain Girbal is Research Engineer at Thales TRT. He earned a master degree in computer science from LRI laboratory at Paris XI University, and, in 2005 a PhD on the subject of Compilers & Software Optimization from Paris XI University. For 3 years, he has been in charge of the Simulation Platform of the HiPEAC European Network of Excellence as an INRIA Futurs engineer. He also participated to various projects including ANR SocLib, SARC IP, UNISIM, and CERTAINTY projects. His research interests include computer architecture simulation methodology, design space exploration methodology, compiler techniques, multi-cores and safety critical systems.

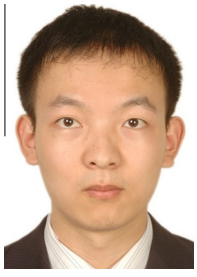
Daniel Goodman is a researcher at the University of Manchester where he is investigating programming models for high productivity languages on chips containing up to 1000 cores. His interests include programming abstractions for multi-core and heterogeneous processors, and GPU computing. Prior to this role he has developed tools and libraries to simplify the programming of novel hardware ranging from heterogeneous CPU-GPU systems to Japan's K super computer. He has a BA and Doctorate in Computer Science from the University of Oxford where he developed the Martlet workflow language for distributed data analysis. His work can be seen at http://apt.cs.man.ac.uk/people/dgoodman.

Behram Khan is a Research Associate in the School of Computer Science at the University of Manchester. He gained his PhD in Computer Science in 2009 from The University of Manchester. Behram's research interests include many-core architectures, and Software and Hardware Transactional Memory. His contact email address is khanb@cs.man.ac.uk.

Souad Koliai received her PhD in Computer Engineering from the University of Versailles (France) in July 2011. Currently she is a post-doctoral researcher at CAPSL at University of Delaware. Her thesis was about developing systematic performance analysis approaches for scientific programs. Her research interests lie at computer architecture, parallel programing, performance analysis, and code optimization.

Being a particularly adroit computer programmer, **Joshua Landwehr** is a former factotum in the field of parallel computing who was employed in the Computer Architecture and Parallel Systems Laboratory at the University of Delaware, where he got his Master in Electrical and Computer Engineering.

Nhat Minh Lê is a PhD student at ENS working on compilers and runtimes for concurrency and parallelization. He graduated from KTH and Grenoble INP Ensimag in 2012, after completing a joint Master's program in computer science.



Feng LI is a PhD student at UPMC and INRIA

Mikel Luján is a Royal Society University Research Fellow in the School of Computer Science at the University of Manchester. His research interests include managed runtime environments and virtualization, many-core architectures, and application-specific systems and optimizations. He has a PhD in computer science from the University of Manchester. Contact him at mikel.lujan@manchester.ac.uk

Avi Mendelson is a professor in the CS and EE departments Technion, Israel. He earned his BSC and MSC degrees from CS, Technion, and got his PhD from University of Massachusetts at Amherst (UMASS) Before joining the Technion, he spent 11 years in Intel, where he served as a senior architect in the Mobile Computer Architectures. He also managed the Academic collaborations of Microsoft R&D Israel for 4 years. His research interests span over different areas such as computer architecture, operating systems, power management, reliability, fault-tolerance, cloud computing, HPC and GPGPU.

Laurent Morin is Founder, Fellow engineer in Compilation technology, and R&D manager at CAPS entreprise. He has worked for the Deutsches Electronen Synchrotron Laboratory (Zeuthen, Germany) and at the IRISA Laboratory where he has made his PhD activities. He has a master degree in Computer Science from the University of Rennes 1.

Nacho Navarro is Associate Professor at the Universitat Politecnica de Catalunya (UPC), Barcelona, Spain, since 1994, and Senior Researcher at the Barcelona Supercomputing Center (BSC), serving as manager of the Accelerators for High Performance Computing group. He holds a PhD degree in Computer Science from UPC. His current interests include: GPGPU computing, multi-core computer architectures, memory management and runtime optimizations. He is a member of IEEE, the IEEE Computer Society, the ACM and the HiPEAC NOE.

Tomasz Patejko graduated from Wroclaw University of Technology in 2009. In 2011 he started working in Barcelona Supercomputing Center.

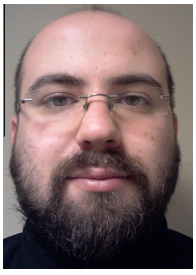
Antoniu Pop is a Lecturer at the University of Manchester. His research interests include high-level parallel programming abstractions, code generation and dynamic optimization. He graduated from ENSIMAG and received his PhD from MINES ParisTech in 2011. He is a member of the ACM.

Pedro Trancoso is an Associate Professor at the Department of Computer Science at the University of Cyprus, which he joined in 2002. He has a PhD and MSc in Computer Science from the University of Illinois at Urbana-Champaign, USA. His research interests are in the area of Computer Architecture and include Multi-core Architectures, Memory Hierarchy, Database Workloads, and High-Performance Computing. He leads the Computer Architecture, Systems and Performance Evaluation Research team CASPER (www.cs.ucy.ac.cy/carch/casper). He has participated in the TERAFLUX EU project, received a 48-core experimental processor from the Intel Corporation, and is a member of the HiPEAC Network of Excellence.

Theo Ungerer is Chair of Systems and Networking at the University of Augsburg, Germany. He is a founding member of the EU network of excellence HiPEAC, and coordinator of the EU projects MERASA (2007–2010) and parMERASA (2011–2014). His research interest include real-time and fault tolerant systems, parallelisation for real-time systems, embedded multi-core hardware design, and organic computing.

Ian Watson is Professor of Computer Science at the University of Manchester, England. His major interest is in the architecture of general purpose parallel computers with particular emphasis on the development of machine structures from an understanding of the requirements of language and computational model issues. His recent interests are in the area of multi-core systems with particular emphasis on approaches to achieve high extensibility using a combination of Dataflow techniques together with transactions to provide a clean approach to introducing the handling of shared state.

Sebastian Weis is a PhD student at the University of Augsburg. His research interests focus on fault-tolerance mechanisms for parallel architectures. He holds a diploma degree in computer science from the University of Augsburg.



Stéphane Zuckerman is a postdoctoral fellow in the Computer Architecture and Parallel Systems Laboratory (CAPSL) at the University of Delaware. His research interests lie in specifying and implementing parallel execution models, as well as high-performance computing performance evaluation and optimization. He received his PhD from the University of Versailles Saint-Quentin-en-Yvelines in 2010.

Mateo Valero is a Full professor at Computer Architecture Department, Universitat Politècnica de Catalunya since 1983. Director Barcelona Supercomputing Center. His research topics are centered in the area of high-performance computer architectures. Published over 600 papers. Served in organization of 300 international conferences. His main awards: Eckert-Mauchly, Harry Goode, "Hall of Fame" member IST European Program, King Jaime I in research, two Spanish National Awards on Informatics and Engineering. Honorary Doctorate: Universities of Chalmers, Belgrade, Las Palmas, Zaragoza, Veracruz of Mexico and Complutense of Madrid. IEEE, and ACM fellow, Intel Distinguished Research Fellow. Member of Royal Spanish Academy of Engineering, Royal Academy of Science and Arts, correspondant academic of Royal Spanish Academy of Sciences, Academia Europaea and Mexicans Academy of Science.