Theme 3 / Design and verification of Systems on a chip architectures
SLS Group: Design

Themes

Architectures and CAD software for multiprocessor systems on chip and networks on chip
Modeling and simulation techniques for software/hardware interfaces
Specification and implementation of embedded systems on chip
Reconfigurable prototyping platform for system validation

Expertise

Scientific
Integration and optimization of Multiprocessor hardware/software systems

Fields of expertise
Multiprocessor architecture, Network on chip, Memory subsystems, On chip embedded operating system, Fast simulation of digital systems, Methods and tools for system level synthesis, Reconfigurable architectures

Know-how
Design of circuits and integrated systems, FPGA, Operating systems, Software/hardware integration

Industrial transfer
Patent for 3D asynchronous circuitry, patent pending on buffer sharing in NoCs, 8 CIFRE theses over the 4 last years (2010 – 2014)

Research keywords

System simulation, network-on-chip, memory hierarchies, 3D design, reconfigurable computing, low-level software, digital system architecture, dynamic binary translation

Contact

Frédéric PÉTROT
GRENOBLE INP Professor – TIMA
(+33) 476 57 48 70
frederic.petrot@imag.fr
Innovative MPSoC Architectures: memory hierarchy, reconfigurable systems

Key-words: HLS tools, dynamic reconfiguration, FPGA visualization,

Members : S. Mancini, O. Muller, F Pétrot, F. Rousseau, A. Abrial, P.H. Horrein, A. Prost-Boucle, Y. Xu, V. Schwambach Costa

Cooperations: Télécom ParisTech, STMicroelectronics, Telecom Bretagne, LIP6

Contracts: iGlance (CATRENE)

Trends in MPSoC are towards massively parallel and homogeneous architectures. The underlying issues of these architectures are the memory wall (access to memory does not scale as Moore’s law), power wall (power consumption is limited by dissipation) and programmability wall (parallel machine are hard to program). To contribute in this field, we provide architectural mechanisms to relieve the pressure on memory, by optimizing memory accesses. We are also working on increasing the efficiency of MPSoC cores by using application specific cores (or hardware components) in reconfigurable systems, with friendly tool-suites for easier programmability. Both research activities are presented in the following sections.

Simplification and optimization of data access

nD-AP Cache : n-Dimensionnal Adaptive and predictive Cache

Pre-fetching in a memory hierarchy is known to alleviate the "memory wall" paradigm but there’s a need to provide strategies more efficient than standard mechanisms such as SPT (Stride Prediction Table). Indeed, there's a need to manage complex memory reference patterns as well as data-dependent ones. The proposed nD-AP Cache performs a stochastic prediction of future references from an analysis of past ones. The prediction mechanism allows to cope with irregular patterns as well as dynamic quasi-random ones. It shows to be more efficient than SPT in many situations. Past work showed the effectiveness of the approach and lead to hardware demonstrators of the nD-AP Cache. Current work focusses on the auto-adaptativity and self tuning of the prediction mechanism in order to precisely set the amount of prefetched data and determine the prefetching distance. The adaptativity is performed according to some dynamically measured characteristic of the fetch sequence as well as the main memory performance (latency, throughput).

Mmopt : Memory Management optimization in a HLS flow

When designing hardware accelerators for low level vision kernels (image correction and transformation), one as to efficiently manage the transportation of data from huge and cheap memories to embedded memories, the closer to the computing unit. Then the designer has to find a trade-off between the amount of loaded data, the size of embedded memories and the overall computing time. To overcome the limitations of standard commercial HLS tools, the Mmopt tool and methodology acts as a pre-process in a HLS flow to help the designer finding pareto optimal solutions and automatically generates an efficient memory management controller. Mmopt targets “non-linear” kernels, which memory references are non-linear functions of loop indexes. The optimization process takes as input the kernel C/C++ code of the Processing Engine (PE) and generates a set of optimal parameters of a generic target Data and Management Controller (DCM). A final code containing both the PE and the DCM is then the entry of the last HLS step.

Past research focused on the proof of concept and benchmarking on a set of kernels. Current research focuses on the parallelization of memory references and on the design of multi-kernel processing engines.

Figure 1: Reconfigurable MPSoC architecture template
Improving reconfigurable system productivity

To provide better computing efficiency, in terms of execution time, power consumption and area in a given technology, MPSoC architectures often integrate application-specific processors (IP). The nature and number of these dedicated hardware components are limited and have to be set up at design time. The reconfigurable computing paradigm gives the opportunity to a virtually infinite hardware component pool.

A reconfigurable MPSoC architecture, as illustrated in Figure 1, is based on homogeneous fabric of cells (chunk of hardware resources, which can be dynamically reconfigured to host a hardware component while the rest of system is still running and computing), reconfiguration controllers (Rctrl manages the relocation of a hardware component from its description in memory to the configuration of a cell) and CPUs. Thus, by time-multiplexing the reconfigurable cells, a reconfigurable MPSoC can tend towards better performance that heterogeneous one, while preserving flexibility.

In spite of their tremendous potential, reconfigurable MPSoCs still fail to convince application programmers. Their design methods and tools offer a poor productivity (time required to arrive to a solution), compared to multi-CPU or GPU solutions, which benefit from standard programming models and better design automation tools. Therefore, we have tackled several locks on the productivity of reconfigurable MPSoC.

The first lock tackled is the generation of hardware component. Currently, High-Level Synthesis (HLS) tools are used to improve designer productivity on reconfigurable fabric (FPGA). However, the existing tools are far from being productive enough: long time to solution, manual iterations in the design flow to obtain a hardware component fitting in a cell (hardware design skills required).

To bring the HLS much closer to the compilation task for CPUs, we have proposed a fast, scalable and standalone methodology, that works under strict resource (and frequency) constraints and produces one unique solution.

We have built a open-source tool, called AUGH (Autonomous and User-Guided High-level synthesis). Experiments conducted on several applications show that our methodology converges rapidly towards a satisfying solution and scales even to complex designs.

The second challenge states in programming reconfigurable MPSoC. Writing applications on these architectures is laborious, poorly portable and hardly scalable to multi-user and/or multi-FPGA systems, mainly because of a mixture of application code (software and hardware components) and flexibility management code.

In order to abstract the complexity for user and thus to ease the integration process in a software-centric application, we have proposed an light-weight and platform-independent Operating System (DNA-OS) extension to handle hardware components (communication, synchronization and reconfiguration). To enhance portability of reconfiguration management procedure, we have also proposed an abstraction layer, called Hardware Component Manager (HCM), which clearly separates the allocation of a hardware components from the control of a reconfiguration procedure. An hardware implementation of the HCM was used in the proposed OS extension.

To compare to integration on other MPSoC platform, we have developed and used the HeRA framework. This framework can abstract implementation and execution methods of a component and currently supports several OS (linux, DNA-OS) and platforms (multi-CPU, GPU, reconfigurable MPSoC).

References
Embedded and low-level software generation

Key-words: Embedded software, device driver, MPSoC, Multi-tile systems, task migration

Members: F. Rousseau, F. Pétrot, N. Fournel, A. Elantably, C. Deschamps, E. Ripert

Cooperations: STMicroelectronics, Magillem Data Service, INFN Roma, CEA LETI

Contracts: SoftSoC (FUI), EURETILE (FP7),

The low level software is the intermediate software layer between the hardware architecture and the software application layer. This low level software, also called Hardware dependent Software (HdS) includes three main parts:

- The Operating System (OS), which manages the sharing of resources. It is responsible for the initialization and management of the application tasks and communication between them. It provides services such as tasks scheduling, context switching, and so on. The relationship with the hardware is tenuous, and mainly through the implementation of drivers.
- The communication layer, which is responsible for managing the I/O operations and more generally the interaction with the hardware components and the other subsystems. This communication layer implements the different communication primitives used for task communication (intra or inter processor).
- The HAL is a thin software layer, which totally depends on the type of processor that executes the software stack, but also depends on the hardware resources interacting with the processor. The HAL performs the processor related accesses on which the device drivers are based to implement the interface for the communication with the device.

To contribute in this field, we have developed a small embedded operating system, called DNA-OS, that has been ported on top of a few typical processors (MIPS, ARM, SPARC, Microblaze, NIOS, x86, ...). This OS serves as basic component for a software design flow able to produce binary code for all computing units of a multiprocessor architecture. The design approach is component-based, static, and very low cost. This design approach was a main contribution few years ago. This allows to specifically tailoring the OS regarding the services required by an application, which corresponds to the constraints that an integrated system has to meet. It is the basic for all the following research activities.

Device driver generation

In terms of communication layer, a set of platform specific device drivers has been developed. However, sharing the device driver development knowledge is complex, and writing drivers is an error prone and a time consuming activity. Therefore, we are currently working on automatic driver generation from high-level hardware and software descriptions. From a practical point of view, this would simplify this task much.

In simplest terms, device drivers are means of communication between the kernel and hardware devices. In more advanced terms, a device driver is a specific type of software component/module in the OS that converts requests from higher-level software (e.g., the kernel or an application program) into a series of low-level input/output (I/O) operations specific to a hardware device (e.g., a network interface controller) abstracting the functionality of a physical or virtual device and managing their operation. It hides completely the details of how the device works.

A device driver interface can be separated into four parts (see Figure 1). The driver requires kernel services like memory allocation, and also offers services (e.g., hardware initialization) to the kernel. b) User application sends general commands to the driver using exported driver interface, while c) libraries provide the driver with some services like string manipulation. d) Lastly, the hardware abstraction layer (HAL) accommodates hardware access methods, which are used by the driver.

One of the most elementary pieces of information about a device and the driver that manages it, is what function the device accomplishes. Different devices carry out different tasks and need information at different semantic level to work out.

TIMA Annual Report 2014 (SLS)
There are devices that play sound samples, devices that read and write data on a magnetic disk, and still other devices that display graphics to a video screen, and so on. Thus, because of the various and interrelated sources of information, driver generation is intrinsically complex.

A method, called Me3D, has been investigated to help in device driver generation. Step by step, information is requested from the designers (IP-XACT model of the platform or device, behavior of the driver, ...) for the final generation. A set of tools is also available, called ADDAX (http://timasls.imag.fr/www/research/addax/addax-setup/). Method and tools are today related to DNA-OS, but will be extended to other OS.

In the context of multi-tile systems connected by NoC, we have provided an original method with the use of a formalization of communication paths to generate drivers. Usually, the way data transfers occur in this architecture is predefined and implicit. Due to the multiple levels of hierarchies, describing explicitly in the architecture model the different stages of data transfers becomes a necessity. A simple way to do it is to enumerate all communication paths between two processing units, and for each communication path all devices involved in the transfer.

Writing one driver per communication path is not conceivable anymore due to the large number of available communication paths in a complex architecture. One solution is to develop generic driver templates that implement several communication paths. For each communication path, a correct template is selected and specialized to efficiently handle the hardware resources. Thus, we obtain a multi-device driver that is then compiled and linked with the operating system and application code to provide the binary code for each processing unit. We provide a communication synthesis process with a good simplicity/control trade-off.

**Task migration in non-SMP architecture**

Task migration is a well known feature in the context of SMP (Symmetrical Multi-Processor) architecture. This becomes very challenging when dealing with non-SMP architecture, and we are developing method and tool to provide such a feature. The main goal is to provide an answer to load balancing, thermal and fault tolerance awareness of processors in MPSoC.

The main idea of task migration is the transfer of a process state plus its address space from the source core (referred to also as home or host node) to the destination one (referred to also as destination node). Its significant cost is coming mainly from the process of transferring the address space. The address space is usually composed of the task stack and heap. However, this is not the only task attribute to be transferred. The whole task state including the address space and the CPU registers, open files and ports, have to be transferred to the destination core to be eventually resumed/ restarted properly.

As a solution, we consider task replication: It consists in having replicas of tasks in the system. When a migration is needed, the task is suspended in its source processor and resumed in the destination processor after the transfer of the process state. The destinations for migrating tasks are determined statically prior to compilation and linking. This enables the system level design process more open for optimum locations of migrating tasks. Although the system level designer must be aware of the sources and corresponding possible destinations of all migrate-able tasks, our solution has been designed so that there is no involvement from the programmer side whatsoever regarding the migration process. Our solution is implemented as a layer between the operating system and the application. It has been chosen to utilize OS application software interface APIs without modifying the operating system itself and that’s to make the solution capable of being plugged over any operating system.

One of the issues of task migration is the inconsistency in the communication arising from the change of the location where the task is supposed to be running in. This consequently, requires informing all the adjacent tasks to the migrating one in the task graph with its new location after migration to keep sending and receiving tokens correctly. Not only location update is important to keep the communication consistent, but also any remaining unprocessed tokens left in channels FIFOs have to be forwarded/re-sent to the right corresponding FIFOs after migration to ensure consistent completion.

**References**


Three-Dimensional Networks-on-Chip

Key-words: 3D-IC, Network-on-chip

Members: F. Pétrot, A. Sheibanyrad, S. Foroutan, M. Bahmani, J. Saade, H. Bel-Hadj-Amor, N. Bentini

Cooperations: ST-Microelectronics, CEA-LETI

Contract: 3DIM3 (Catrene)

As the end of scaling CMOS transistors comes in sight, the 3D-Integration may come to the rescue of the industry to allow for a continuing exponential growth of integration during the 2015-2025 periods. Moreover the incorporation of the third dimension in the design of the NoCs allows the exploitation of three dimensional topologies which result in a major improvement in network performance.

Through-Silicon-Via (TSV) has the potential to offer a great vertical interconnect density and features an extremely small inter-wafer distance. Although 3D-Integration using TSVs introduces a whole new set of application possibilities, it also introduces new architecture level design issues. Fabricating a 3D integrated circuit using TSV technology involves several extra and costly manufacturing steps, and each extra step adds a risk for defects, resulting in potential fabrication yield reduction. Additionally, the TSV interconnect pitch (mainly due to its pads) imposes a larger area overhead than the corresponding horizontal wires and a TSV consumes all layers in the upper die in addition to the top layer in the lower die.

To deal with this cost-efficiency trade-off, we had suggested reducing the number of vertical links to be exploited by the network. However despite the undeniable benefits of reducing the number of TSVs, removing vertical links has a degrading impact on the NoC performance. Our major research efforts during 2013 focused on how we can optimally assign elevators (vertical links) to nodes of a Vertically Partially Connected 3D-NoCs.

Assignment of Vertical Links (Elevators) to Nodes of Vertically-Partially-Connected NoCs

While the number and location of vertical links are determined at design-time according to the hardware constraints, the major remaining challenge is to assign to each router it’s up and down elevators according to target application. This consists of assigning to each router at configuration-time, the address of a router on the same die that possesses a vertical link to the upper die, as the up-elevator, and the address of a router on the same die that possesses a vertical link to the bottom die, as the down-elevator. We decouple the (fixed) hardware decision of number and location of elevators that must be solved at design-time, from the (flexible) elevator-assignment that can be reconfigured later according to application requirements. The main advantages of this decoupling are: 1) The elevator number/location decision can remain a completely hardware-dependent and application-independent problem. 2) Without changing available hardware resource, we can optimize the NoC performance for given application requirements, just by configuring elevator assignment. 3) When a vertical link fails (followed by a TSV failure), we can reconfigure the assignment to avoid routing through the failure link by maintaining the optimal performance. Different elevator-assignments for the same number/location of elevators (the same topology), can change the network performances.

For solving the assignment optimization problem, we have proposed a heuristic algorithm based on tabu search. To do this we have (1) shown the relation between link buffer utilization, network performance (in term of network saturation threshold), and elevator assignment, (2) defined a novel quantitative metric to determine which assignment leads to a better network performance, and (3) proposed a straightforward approach to choose the neighborhood solution in each step of search.

We use the link buffer utilization to quantify the link-level local traffic. The link with the highest buffer utilization is the bottleneck of the network. Buffer utilization is a strong weight metric for our optimization purpose as it reflects in the same time both link (flit) rate and link congestion. When packets are in contention they become stalled and their flits are accumulated in link buffers. If the flit rate is high it means more flits are accumulating and so the buffer utilization becomes larger. Different elevator assignments lead the global traffic routed via different paths, which consequently lead to different local traffic distributions, which finally lead to different performance results. Since there are fewer elevators than nodes (routers) on each die, several nodes are assigned to a single elevator. An inappropriate assignment may thus result in creation of local hotspots which significantly deteriorates the global performance. In order to avoid an assignment leading to such an imbalanced and sharp traffic distribution, our algorithm detects the bottleneck. Then, by proposing new assignments it tries other local traffic distributions and verifies if the bottleneck
appeased and thus the saturation threshold can be improved. If not, then the algorithm checks whether the second bottleneck (i.e. the link with the second highest buffer utilization) can be improved, and so on to the least congested link (i.e. the link with lowest buffer utilization). So for any given global (i.e. end-to-end) traffic distribution, the algorithm searches for an elevator assignment that leads to a smoother local traffic distribution and thus optimizes the NoC saturation threshold.

Since the solution-space is huge, we have to stop the algorithm after a maximum number of attempts (i.e. when the number of tabu assignments reaches a maximum). However, depending on the initial solution, the algorithm may converge to a local optimal solution and thus cannot progress anymore, before achieving the intended maximum number of attempts. We cannot claim that the final solution found by the algorithm, is the absolute optimal one. Local search approaches (and specifically tabu search according to [36]) can obtain near optimal solutions. These solutions depend on the initial assignment.

Figure 1 shows how significantly the final assignments obtained after applying the proposed algorithm to a set of arbitrary assignments improve the network performance. The figure shows an improvement of about 100% in saturation threshold for each configuration. When the maximum number of tabu assignments evolves from 50 to 100, 500, 1000, 5000, 10000, or 100000, the algorithm has more time to explore the solution-space and find better solutions. However, as the algorithm converges very rapidly to the near optimal it can find, a much higher number of attempts does not necessarily provide a much better solution. The results for the cases of 5000, 10000, and 100000 (or more precisely 21472 as after that the algorithm cannot progress anymore) are almost the same.

In order to give an idea about the optimization complexity, Table 1 compares the algorithm run time under Linux on a PC with Intel Xeon W3530 at 2.8 GHz and with 12 GB of RAM, for different maximum number of attempts, and for a random and a greedy nearest initial assignments. The optimum solution with greedy initial assignment is obtained more than 9 hours sooner than that of random assignment.

<table>
<thead>
<tr>
<th>Random</th>
<th></th>
<th>One of the Nearest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabu Number</td>
<td>Hop Count</td>
<td>Run Time</td>
</tr>
<tr>
<td>0</td>
<td>7.56</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>7.52</td>
<td>35s</td>
</tr>
<tr>
<td>100</td>
<td>7.50</td>
<td>64s</td>
</tr>
<tr>
<td>500</td>
<td>7.44</td>
<td>263s</td>
</tr>
<tr>
<td>1000</td>
<td>7.39</td>
<td>916s</td>
</tr>
<tr>
<td>5000</td>
<td>7.23</td>
<td>6662s</td>
</tr>
<tr>
<td>10000</td>
<td>7.13</td>
<td>16698s</td>
</tr>
<tr>
<td>21472</td>
<td>7.08</td>
<td>48127s</td>
</tr>
</tbody>
</table>

References
[1] Sahar Foroutan, Abbas Sheibanyrad, Frédéric Pétrot, Assignment of Vertical Links to Routers in Vertically-Partially-Connected 3D-NoCs, IEEE Transactions on CAD (accepted with a major revision)
Methods and tools for computer aided design

**Key-words:** simulation, debug, profiling

**Members:** F. Pétrot, N. Fournel, D. Hedde, M.-M. Hamayun, L. Michel, S. Lagraa, M. Gligor, M. Cunha, Th. Ducroux

**Cooperations:** STMicroelectronics, Magillem Data Service, Kalray

**Contracts:** SoCTrace (BGLE), Acose (BGLE), COMCAS (CATRENE), Kalray

The CAD activities mainly covers the following themes: Modeling, Simulation and Debug and Profiling.

**Fast simulation strategies**

Simulation of large scale integrated, potentially heterogeneous, multiprocessor platforms is a long lasting theme of the SLS group. This work is of primary necessity as the number of processors in integrated circuits is raising, and therefore the simulation times are increasing constantly. As predicted by ITRS the evolution of integrated system architecture tends to embed several hundreds of processors, turning multi processors systems into many-core systems. These systems are mainly based on VLIW processors architectures (e.g. Tilera TileGx72 or Kalray MPPA256) to reduce the complexity of the processors to integrate a larger number of them. On top of that, these many-core systems rely on a Network-On-Chip based interconnect. As a result, the execution of the software on Instruction Set Simulators during simulation (making the processor the ultimate hardware/software interface) is not viable anymore. Several innovative approaches have been defined and demonstrated.

For several years, the so called native simulation approach has been seen as the implementation of the low level software functions of an Operating System using a simulator API. This has become specifically popular with the wide dissemination of SystemC, as it provides many API functions similar to the ones of an OS. The idea is thus to model the system as a stack of layers, and each upper layer can rely on the functions provided by the lower layer layers. If the abstraction is coarse, then several layers can be merged to provide at a low execution cost the functions (usually realized on top of an operating system or a high level hardware abstraction such as SystemC TLM), if the abstraction is fine grain, then the "real" realization (some OS code for example) is provided, and the hardware can be described more accurately at the cycle level.

All these level of abstractions share the same issue in native simulation, the address spaces. Indeed, there are two address spaces merged in one, the one from the target system and the one from the simulator.

Some solutions have been proposed in the recent years to overcome this issue, but they fix some development constraints on the application and the operating system running on the target system. Among these solutions one propose to use the simulator address space during the application by inserting some translations at simulation start.

Our main contribution in native simulation has been to define an approach that tackles this problem of address spaces by using widely spread hardware solution. Desktop computer processors embed hardware to assist virtualization and protect the host operating system from the guest operating system. This hardware allows to make a fresh complete address space for the guest operating system and translate it to the host address space transparently. The guest operating system is then unaware of the fact that it is running in a virtual environment. By placing the target system as the guest operating system, the problem of conflicting address spaces raised earlier is solved.
As far as processor simulation is concerned, the use of dynamic recompilation techniques is very promising from a speed perspective. The Dynamic Binary Translation of VLIW architectures on scalar architectures is still an open ground. Our contribution in this topic is to propose a solution to tackle the issues. Among these issues, we can cite the parallel execution on instructions, expressed in the binary code for the VLIW architecture while sequentially is the base of scalar processors code. The write back of registers is the key of this issue. We proposed a solution based on a single assignment register representation in the translation which allows tackling this issue and other issues raised by these architectures.

**Debug and Profiling**

The emergence of massively parallel embedded systems due to the arrival of the multi- and many-core integrated systems opened new issues about debugging and profiling applications. Debugging and profiling in that context cannot be considered as usual debug, *i.e.* launching a debug process per actual process to follow the flow of computation. As there may be hundreds of processes cooperating, the real problem is to identify wrong behaviors like wrong sharing, race conditions, inappropriate understanding of the consistency issues or bad memory accesses sequences due to bad memory placement.

![Figure 1: Interleavings of memory accesses, the black dot represent the synchronizations](image)

Our proposition in this field is based on simulation and execution traces. This infrastructure has the advantage to be be non-intrusive, *i.e.* not modifying timing nor behavior of the traced models, and allow to collect information not available in other cases.

As far as identifying wrong behavior like problem of consistency or race conditions is concerned, a formal approach would be ideal, but unfortunately it is clearly not possible to handle actual programs with these techniques. So we focus on a more practical approach in which we analyze execution traces describing ordered communications operations per processor.

For race condition analysis, these operations are merged per process from the partial orders of the traces. Despite the use of these partial orders, the analysis is trace based, a correct execution can still be detected as leading to a potential race condition. The detection algorithm is exponential, but behaves well in practice if the processes synchronize often, as these synchronizations are indeed limiting the number of events to be considered in parallel. *Figure 1* illustrates the intervals that can be built from a trace.

Virtual prototyping allows releasing constraints on what can be verified today. Indeed, the consistency check on actual parallel machines relies on the fact that, among others, each write must be done with a different value in order to be able to know what process or processor produced it. NP completeness of these checks can possibly (it is still to be proved) be made polynomial by adding information that the virtual prototype may be able to produce. Up to now, we have been working on sequential consistency, but we plan to extend the approach to other memory consistency models.

As far as profiling is concerned, the mass of information collected using traces makes difficult the identification of weak points. A successful collaboration with colleagues from the computer sciences lab of Grenoble to apply data-mining techniques to automatically identify repeated or costly patterns in the traces. Interesting results have been obtained on both performance and power profiling experiments, for example with the automatic detection of high latencies of floating point computation function.

To summarize, we apply here existing data-mining strategies to our traces to detect the occurrence of events that occur either very often or very rarely, and then have the designer analyze himself the identified points. We believe that due to the specific structure of the traces, more efficient algorithms than the general purpose ones can be proposed.

**References**

