(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 March 2022 (17.03.2022)

WIPO | PCT

(10) International Publication Number
WO 2022/053183 A1

(51) International Patent Classification:
G06F 9/30 (2018.01) *G06F 9/38* (2018.01)

(21) International Application Number:
PCT/EP2021/054748

(22) International Filing Date:
25 February 2021 (25.02.2021)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
20195570.5 10 September 2020 (10.09.2020) EP

(71) Applicant: **NEC LABORATORIES EUROPE GMBH** [DE/DE]; Kurfürsten-Anlage 36, 69115 Heidelberg (DE).

(72) Inventor: **THUERCK, Daniel**; NEC Laboratories Europe GmbH, Kurfürsten-Anlage 36, 69115 Heidelberg (DE).

(74) Agent: **ULLRICH & NAUMANN**; Schneidmühlstraße 21, 69115 Heidelberg (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) **Title:** METHOD AND SYSTEM FOR SUPPORTING THROUGHPUT-ORIENTED COMPUTING

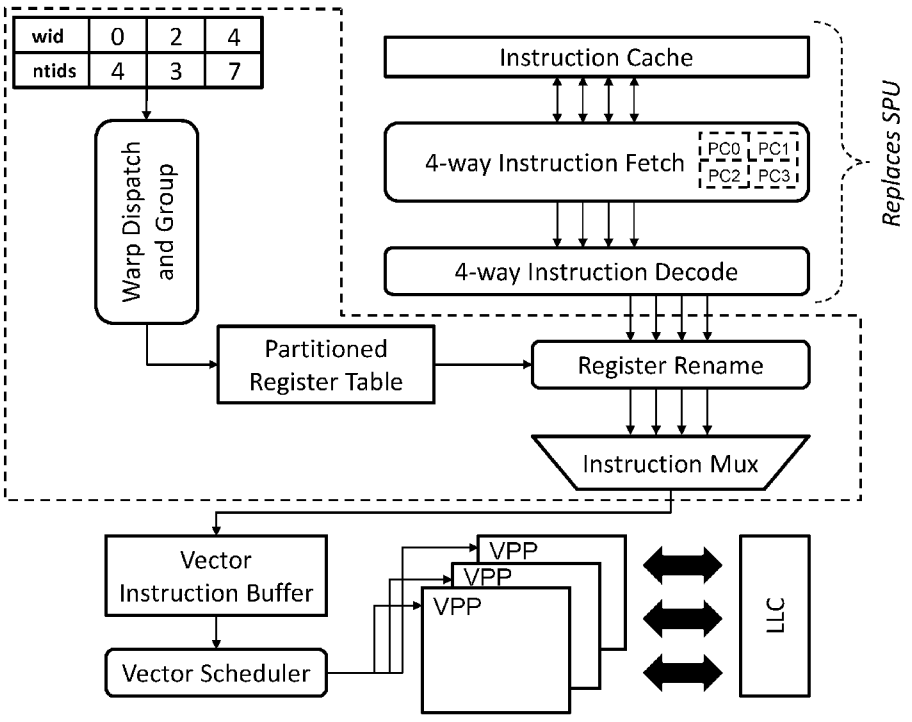


Fig. 5

(57) **Abstract:** A method for supporting throughput-oriented computing, in particular in a high- performance computing system, wherein a single instruction multiple threads, SIMT, program is configured to launch warps, wherein each warp comprises 5 threads to be executed in lockstep within their warp, wherein the warps' individual warp sizes are used as a runtime parameter for the SIMT program, such that a parameterized SIMT program is provided, which is parameterizable via the warp sizes, and wherein the parameterized SIMT program is executed on a single instruction multiple data, SIMD, vector architecture. Furthermore, a corresponding system is disclosed.

[Continued on next page]

WO 2022/053183 A1

Published:
— *with international search report (Art. 21(3))*

METHOD AND SYSTEM FOR SUPPORTING THROUGHPUT-ORIENTED COMPUTING

The present invention relates to a method for supporting throughput-oriented computing, in particular in a high-performance computing system.

Furthermore, the present invention relates to a system for supporting throughput-oriented computing.

The last decade has seen a massive increase in raw compute power due to the inclusion of graphics processing units (GPUs) in high-performance computing (HPC) systems. GPUs augment the previously dominant multi-core CPU (central processing unit) setups by adding the option to effectively perform throughput-oriented computing (TOC). The non-patent literature of *M. Garland and D. B. Kirk, "Understanding throughput-oriented architectures", Communications of the ACM, 53(11):58–66, 2010* mentions three important design principles for such systems: an abundance of rather simple processing units, SIMD-style execution and hardware multithreading. In this spirit, the scheduling hardware on GPUs is kept simple in favor of including more local memory and execution units.

NVIDIA's GPUs are primarily accessed by the native CUDA API, implementing a 2-level, regular grid of work items of constant size as its native programming model. The effective use of the hardware mandates an abundance of work items, outnumbering the available hardware compute units by far. Work items that are actively being computed by the compute units in a given moment may be also designated as threads in this context. On such GPUs, 32 work items/threads are jointly executed in SIMT style inside a so-called warp, with multiple fixed-size warps sharing a shared multiprocessor (SM) in order to hide latencies. Consequently, architectures implementing the TOC principles perform poorly when encountering control flow irregularity (e.g. branch divergence, fine-grained synchronization) or data irregularity (e.g. differing resource requirements between work items/threads).

Opposed to that, modern multi-core CPUs are optimized for latency and can use out-of-order execution to reorder instructions and speculative execution to combat control flow irregularity. Furthermore, they support simultaneous multithreading (SMT) to swap between executing applications, albeit at much higher cost than hardware multithreading in GPUs. Similarly, all their CPU cores are independent and request resources as needed. The concepts of latency and throughput-oriented architectures are diametrical opposites.

Accordingly, the last decade has seen continued exponential performance increases in HPC systems, well after the predicted end of Moore's Law for CPUs, however, largely due to the widespread adoption of throughput-oriented compute accelerators such as GPUs. When faced with irregular yet throughput-oriented applications, their simple, grid-based computing model turns into a serious limitation. Indeed, classical domains where TOC proves effective are now facing the increasing use of irregular applications: e.g., sparse matrices, graph neural networks and sum-product networks. Sparse matrices are the bedrock to modern applications in computing science and the machine learning community has recently started to adopt new and irregular architectures such as graph neural networks and sum-product networks.

The architectural continuum between throughput-oriented SIMT/SIMD designs and latency-based (e.g. SMT) designs has been explored to great depths. Multiple extensions to SIMT designs have been proposed: Scalar co-processors for GPU cores that avoid repeated scalar computations; central schedulers share the GPU between host threads or dynamic re-grouping of threads from a warp or block into convergent subgroups. Similar to SMT context switches, the non-patent literature of *S. Frey, G. Reina, and T. Ertl.: "SIMT Microscheduling: Reducing Thread Stalling in Divergent Iterative Algorithms", In PDP'20* propose a model for oversubscription of tasks to SIMT cores and a method for faster context switches using the cores' L1 cache. Similarly, work-stealing between warps has been explored. On the other end of the spectrum, multiple works have investigated layering a SIMT scheduler on top of arrays of in-order CPU cores. These arrays switch between MIMD mode (each processor operates independently) and SIMT mode (control logic is shared by all processors) to save power. Subsequent works

extend this idea into a form of “hardware auto-vectorization” of scalar code. In order to group similar instructions on cores of the array, expensive crossbars are required.

Liquid SIMD (cf. non-patent literature of *N. Clark, A. Hormati, S. Yehia, S. Mahlke, and K. Flautner, Liquid SIMD: “Abstracting SIMD hardware using lightweight dynamic mapping”, In HPCA’07*) and Vapour SIMD (cf. non-patent literature of *D. Nuzman, S. Dyshel, E. Rohou, I. Rosen, K. Williams, D. Yuste, A. Cohen, and A. Zaks, “Vapor SIMD: Auto-vectorize once, run Everywhere”, In CGO’11*) on the software side and ARM’s SVE hardware extensions (cf. non-patent literature of *A. Armejach, H. Caminal, J. M. Cebrian, R. González-Alberquilla, C. Adeniyi-Jones, M. Valero, M. Casas, and M. Moretó, “Stencil codes on a vector length agnostic architecture”, In PACT’18*) improve SIMD systems for irregular applications: they offer a convenient way to set the SIMD vector length at runtime, adapting to tasks with varying resource requirements.

In view of the above, it is therefore an objective of the present invention to improve and further develop a method and a system of the initially described type for supporting throughput-oriented computing in such a way that handling irregularity, in particular data and/or control flow irregularity, on throughput-oriented processors is improved.

In accordance with the invention, the aforementioned objective is accomplished by a method for supporting throughput-oriented computing, in particular in a high-performance computing system, wherein a single instruction multiple threads (SIMT) program is configured to launch warps, wherein each warp comprises threads to be executed in lockstep within their warp, wherein the warps’ individual warp sizes are used as a runtime parameter for the SIMT program, such that a parameterized SIMT program is provided, which is parameterizable via the warp sizes, and wherein the parameterized SIMT program is executed on a single instruction multiple data (SIMD) vector architecture.

Furthermore, the above mentioned objective is accomplished by a system for supporting throughput-oriented computing, the system comprising a programming

model and a SIMD vector architecture, wherein the programming model is configured to provide a SIMT program for launching warps, wherein each warp comprises threads to be executed in lockstep within their warp, and to use the warps' individual warp sizes as a runtime parameter for the SIMT program, such that a parameterized SIMT program is provided, which is parameterizable via the warp sizes, and wherein the SIMD vector architecture is configured to execute the parameterized SIMT program.

According to the invention it has first been recognized that when faced with irregular yet throughput-oriented applications, a simple grid-based computing model of throughput-oriented compute accelerators such as GPUs turns into a serious limitation. Instead of repeatedly tackling the issues of irregularity on the application layer, it has been further recognized that a generalization of a SIMT (single instruction multiple threads) model (such as the CUDA model) to irregular grids can be supported through modifications to already established throughput-oriented architectures. To that end, is has been implemented a unifying approach – adhering to the SIMT principles – based on an unlikely ally: a SIMD vector architecture. Consequently, a SIMT program is configured to launch warps, wherein each warp comprises a package of threads to be executed in lockstep within their warp. Thus, the threads inside a warp are operating in lockstep. According to the invention, the warps' individual warp sizes are used as a runtime parameter for the SIMT program, such that a parameterized SIMT program is provided, which is parameterizable via the warp sizes. By doing this, the warps are directly exposed to users. The parameterized SIMT program is executed on a single instruction multiple data (SIMD) vector architecture.

Thus, the present invention provides a method and a system for supporting throughput-oriented computing, wherein handling of irregularity, in particular data and/or control flow irregularity, on throughput-oriented processors is improved.

The term "SIMD architecture" may be understood, in particular in the claims, preferably in the description as a term applied to architectures that execute a single instruction on multiple data items simultaneously. For instance, a corresponding definition is given in the non-patent literature of Flynn, Michael J,

"Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers C-21(9): 948-960, 1972. Modern CPUs may implement this principle in the form of vector registers, containing multiple scalar values in a single (hardware) vector register. For example, in Intel's AVX extension, usual sizes are 8 to 16 scalars contained in a single hardware vector register. Vector registers comprising far more than this number of scalar values may be designated as wide-SIMD. Thus, for example, a wide-SIMD architecture may have at least 32 scalars. An example is NEC's Aurora architecture with 256 scalars. Each scalar slot of a vector register may be referred to as a lane.

However, a SIMT architecture considers each lane of a SIMD registers as a separate program, designated as a thread. It builds hardware support for potential divergent execution within a SIMD register through masking out single lanes. Furthermore, the resulting latencies are hidden through hardware multithreading. In NVIDIA's SIMT architectures, 32 threads are simultaneously executed in lockstep as a warp.

According to embodiments of the invention, a programming model may be provided, wherein the programming model is based on a SIMT programming model, in particular based on the CUDA programming model. The programming model may be extended for handling data irregularity by having a user input a list of warp sizes. Traditionally, CUDA kernels are parameterized over a grid of blocks, e.g.

```
kernel<<<m, n>>>(...);
```

which launches `m` blocks with `n` threads each. On the hardware, each block is scheduled onto a streaming multiprocessor (SM). However, the block abstraction of the CUDA programming model hides the underlying execution model: Inside a block, threads are executed in packages on 32, which may be designated as warps. All threads in a warp operate in lockstep and branches or conditionals are implemented by predicating the execution in some threads; this corresponds to the SIMT model as mentioned above. In recent GPU architectures, all threads in a warp access the same partition in the streaming multiprocessor's (SM's) register

file and communicate through it with low-latency. Blocks of variable sizes m_i can only be emulated by setting the block size to $m = \max\{m_i\}_i$ and masking out threads in each block. According to embodiments of the invention, it has been gotten rid of the block abstraction and directly exposed warps to users. In order to handle data irregularity, the individual warps' sizes have been made a runtime parameter. Hence, instead of passing parameters `m,n` to the kernel, this requires passing an explicit list of warp sizes:

```
const int w_list[] = {4, 11, 3, 2, 8, 3};
kernel<<<m, w_list>>>(...);
```

According to embodiments of the invention, it may be provided that the warps are distributed to SIMD vector cores of the SIMD vector architecture. Advantageously, it may be provided that the warps are distributed by a round-robin method to the SIMD vector cores of the SIMD vector architecture. Thus, a fair distribution of the warps over the vector cores can be achieved.

According to embodiments of the invention, the SIMT program may be mapped onto SIMD vector registers of the SIMD vector architecture by using predetermined (explicit) vector registers of the SIMD vector architecture. The predetermined vector registers are initialized by SIMT metadata of SIMT metadata registers. Thus, explicit (hardware) vector registers are used to have a link to the SIMT metadata. Hence, a method can be provided to execute scalable SIMT code with minimal modifications in different runtime configurations on SIMD processors, i.e. SIMD vector cores, using the metadata registers.

The term "metadata" may be understood, in particular in the claims, preferably in the description as parameters that identify a thread. Thus, metadata may comprise parameters that identify a thread in the SIMT context. Hence, e.g., the index of the work item/thread within the global grid may be considered as metadata. In SIMT architectures, each thread stores this metadata, which is stored and handled by the warp scheduler managing the hardware multithreading. According to embodiments of the invention, this metadata may be stored in pre-determined SIMD vector registers instead. Thus, each kind of metadata may be mapped onto

one vector register, wherein each lane represents one thread (work item) and the mapping between a thread inside a SIMT warp and lane in the metadata register is fixed.

According to embodiments of the invention, the SIMT metadata may include a thread index information, a warp index information and/or a warp size information. Thus, thread indices, warp indices and/or warp sizes can be used in order to handle data irregularity, wherein an execution of SIMT code on SIMD architectures is enabled by concatenating threads' registers into SIMD registers.

According to embodiments of the invention, a mapping of the threads' SIMT metadata registers to lanes in SIMD vector registers of the SIMD vector architecture may be performed at runtime of the SIMT program. Thus, it is followed a SIMT-on-SIMD paradigm by mapping the corresponding registers of threads in a warp to lanes in SIMD registers. Rather than at compile time, that mapping is performed on runtime. Thus, the handling of data irregularity can be achieved efficiently.

According to embodiments of the invention, the SIMT program may be compiled to an intermediate code representation, wherein the intermediate code representation is configured to replace one or more identifiers pertaining to the SIMT program's runtime configuration with references to SIMT metadata registers. The SIMT execution can be emulated by explicitly associating keywords (such as `tid` for a thread's index) in metadata registers to each lane, such that a parameterized SIMD execution is provided. Appearances of those keywords in the code can be then translated to registers with suitable execution. Unless there is branching involved, SIMT code can translate 1:1 into SIMD code. A predicated instruction can be realized through masks over SIMD vector registers. For architectures that support setting a vector length (`vl`) at runtime, a scalar metadata register may be added. Such architectures, commonly called "vector processors", operate on vector registers in a way such that the latency of an instruction is determined by the number of scalar date items in its vector registers. The latter is set as `vl`, thus resulting in shorter runtimes where parts of vector register are not filled with data.

Due to the lack of per-thread program counters (PCs), SIMD hardware is unable to track execution paths of different threads. Instead, all threads must follow the same path of execution. Inactive threads' lanes are masked out in accordance with an embodiment of the invention. In order to simulate this behavior, a compiler may carry out two steps: first, whenever branching (through e.g. loops) is involved, create a predicate indicating whether a thread took the branch or not. The resulting predicate is then applied to all instructions that can potentially follow. If additional predicates appear, they should be combined via `and`. Additionally, a loop must be repeated unless its condition evaluates to `false` for all threads in a warp. Thus, according to embodiments, warp-synchronous branching commands may be provided, which only branch if all threads in the warp agree on a predicate's result.

According to embodiments of the invention, the intermediate code representation may include a scalar branch control such that warp-synchronous branching commands are provided that only branch if all threads in the warp agree on a predicate's result.

According to embodiments of the invention, a predicate may be created whenever branching is involved in the SIMT program. The predicate indicates whether a thread took a branch or not, wherein the predicate's result is applied to all instructions that can (potentially) follow implicitly without further annotation. Thus, the predicate's result can be applied automatically to the instructions.

According to embodiments of the invention, a control flow of the SIMT program may be modified to emulate branching by using a stack of masks for SIMD vector registers, wherein a translation from thread-wide branching instructions to warp-wide branching instructions is provided. Furthermore, it may be provided that whenever a branching instruction is encountered, the corresponding predicate is evaluated and the branch is taken only if all of the involved threads evaluate the predicate to `false` (cf., e.g., an "`allbra`" instruction as illustrated in Fig. 1 at (2)).

According to embodiments of the invention, it may be provided that a partitioning scheme is implemented for executing multiple partitions on a SIMD vector core of the SIMD vector architecture, wherein a partition includes one or more warps to be executed on the SIMD vector core.

According to embodiments of the invention, several warps included in a partition may be packed into a single SIMD vector register by concatenating their metadata, wherein a vector length of the SIMD vector register is increased accordingly. Thus, an active vector length management is performed.

According to embodiments of the invention, a register renaming may be performed inside SIMD vector cores of the SIMD vector architecture in order to multiplex instruction streams from multiple partitions into a joint vector instruction buffer. Thus, each SIMD vector core can have one vector instruction buffer.

According to embodiments of the invention, the register renaming may be performed based on a reprogrammable partitioned register table.

Further features, advantages and further embodiments are described and may be become apparent in the following:

In order to bring the success and simplicity of CUDA's programming models to applications dealing with data and/or control flow irregularities, a method and a system in accordance with embodiments of the invention provide an extension that models the applications' inherent data irregularity. In keeping with the SIMT spirit, arising issues can then be solved with minimal hardware involvement. In fact, embodiments of the invention can show that the classic architecture of vector computers (which may be designated as SIMD computers or "wide-SIMD" computers) such as NEC's SX-Aurora TSUBASA have all the components that a design in accordance with an embodiment of the invention requires. Thus, a modified vector architecture is proposed that exploits the on-chip register renaming unit to support both SMT and SIMT execution efficiently on one (wide-)SIMD chip. As a result, a design in accordance with an embodiment sets itself apart from others by offering both a simple programming model and incremental

(and thus, quickly realizable) hardware modifications that are consistently designed around the support for irregular applications.

According to embodiments of the invention, CUDA's PTX ISA may be extended to support an irregular compute model. The compute model can be mapped to wide-SIMD processors in order to handle data irregularity.

According to embodiments of the invention, it may be provided that register renaming is used as a tool to emulate SIMT-like hardware multithreading on a simple vector core, avoiding costly context switches, such that a handling of control flow irregularity is achieved.

According to embodiments of the invention, approaches for handling data irregularity and for handling control flow irregularity may be integrated into a modified SX-Aurora architecture.

Embodiments of the invention extend the commonly used, regular computing model in the area of throughout-oriented computing to tasks that exhibit data and control flow irregularity (which may be referred to as "irregular tasks"). Embodiments of the invention offer ways to combine and map instances of throughput-oriented programs ("kernels") to a conventional, regular wide-SIMD processor.

By combining the techniques outlined above, embodiments of the present invention require only minor, incremental modifications to existing hardware designs in order to extend their capabilities to irregular computing. Thus, the invention adheres to the principles of throughput oriented execution and SIMT execution while integrating some techniques from the (orthogonal) concept of latency-based computing to utilize the underlying hardware as much as possible. This guarantees simple inclusion into existing, well-supported HPC stacks and widespread adoption in the community.

At the same time, embodiments of the invention allow the use of a simple, batched programming model that operates on the same principle as SIMT kernels do;

kernel developers do not need to acquire a new skill when using hardware modified according to principles in accordance with embodiments of the invention.

Embodiments of the invention may be purely targeted at processors and kernels on throughput-oriented systems. Modifications in accordance with an embodiment of the invention do not turn wide-SIMD systems into performant general-purpose systems. Accordingly, it may be provided that the limitations from throughput-oriented systems such as GPUs are inherited: best performance is only reached for well-parallelized code with high arithmetic intensity, low memory intensity and ideally, only a limited amount of branching.

- As NEC's SX Aurora Tsubasa product is an example of a wide-SIMD card, the proposed modifications can immediately be implemented there, preparing it for more irregular throughput-oriented applications that are currently popping up (such as graph neural networks in machine learning).

- The simple programming interface and SIMT-to-SIMD conversion can largely be implemented in software, including for regular programs. Techniques from embodiments of the invention may be included into NEC's NCC compiler in order to port CUDA code to Aurora. After the hardware changes, NEC's Aurora can be a HPC accelerator to offer native support for SIMD, SIMT and SMT principles.

Hence, embodiments of the present invention relate to methods and systems to modify wide-SIMD HPC accelerators (e.g., but not exclusively, NEC's SX-Aurora Tsubasa) in order to improve execution of irregular throughput-oriented programs. Embodiments may propose an integrated system of compiler extensions and modifications to hardware components that result in low-overhead methods for scheduling irregular applications using a simple interface. Embodiments of the invention offer a path towards bridging the gap between throughout oriented computing and the demands of irregular applications by incremental changes to existing solutions.

Embodiments of the invention may describe a method for processing parameterized SIMT programs suffering from data and control flow irregularity efficiently on SIMD hardware with minimal hardware changes using an integrated system of programming model, compiler extension and re-purposing of existing hardware. The method offers ways of merging differently parameterized programs into a single instruction stream in order to fully utilize the underlying hardware.

There are several ways how to design and further develop the teaching of the present invention in an advantageous way. To this end it is to be referred to the patent claims subordinate to patent claim 1 on the one hand and to the following explanation of further embodiments of the invention by way of example, illustrated by the figure on the other hand. In connection with the explanation of the further embodiments of the invention by the aid of the figure, generally further embodiments and further developments of the teaching will be explained.

In the drawing

Fig. 1 is a code example illustrating a compilation pass from CUDA to vector-ready PTX for a method in accordance with embodiments of the invention, wherein a sparse matrix-vector multiplication (SpMV) kernel is used,

Fig. 2 is a schematic view illustrating an example of distributing warps to SIMD vector cores in accordance with embodiments of the invention,

Fig. 3 is a schematic view illustrating a register renaming process in accordance with embodiments of the invention,

Fig. 4 is a schematic view illustrating an architecture overview of NEC's SX-Aurora TSUBASA,

Fig. 5 is a schematic view illustrating an architecture overview of a vector core for a system in accordance with embodiments of the invention,

Fig. 6 is a schematic view illustrating a method in accordance with an embodiment of the present invention,

Fig. 7 is a schematic view illustrating a process of metadata register programming for processing warps and partitioning of multiple warps into a single partition,

Fig. 8 is a schematic diagram illustrating simulation results from a model-driven simulation of a setup in accordance with an embodiment of the invention,

Fig. 9 is a schematic diagram illustrating further simulation results from a model-driven simulation of a setup in accordance with an embodiment of the invention, and

Fig. 10 is a schematic diagram illustrating further simulation results from a model-driven simulation of a setup in accordance with an embodiment of the invention.

Fig. 1 shows a code example illustrating a compilation pass from CUDA to vector-ready PTX for a method in accordance with embodiments of the invention, wherein a sparse matrix-vector multiplication (SpMV) kernel is used. According to the embodiment as illustrated by Fig. 1, a solution is provided that integrates a modified compilation pass from CUDA (cf. C source code on the left side of Fig. 1) to vector-ready PTX (cf. the middle of Fig. 1), an extended version of NVIDIA's PTX that enables execution of SIMT code on SIMD architectures by concatenating threads' registers into SIMD registers. To that end, (1) special registers for warp and thread indices are added, (2) thread-individual branch statements are interpreted as to voting functions over the whole warp and (3) a stack of masks for SIMD registers inside a loop is maintained. On the right side of Fig. 1, a corresponding exemplary process is illustrated for a warp having four threads.

PROGRAMMING MODEL OF AN EMBODIMENT

According to embodiments of the invention, a programming model can be employed, which is based on a SIMT programming model. To that end, a generalization of CUDA's grid-based compute model to irregular workloads is proposed. Traditionally, the CUDA kernels are parameterized over a grid of blocks, e.g.

```
kernel<<<m, n>>>(...);
```

which launches m blocks with n threads each. On the hardware, each block is scheduled onto a streaming multiprocessor (SM). However, the block abstraction hides the underlying execution model: Inside a block, threads are executed in packages on 32, which can be designated as warps. All threads in a warp operate in lockstep and branches or conditionals are implemented by predicating the execution in some threads; this is commonly referred to as the SIMT model. With thread-independent scheduling having been introduced in the Pascal microarchitecture, the lockstep model has been somewhat relaxed. Each thread now has its own program counter (PC), which allows the warp scheduler to interleave instructions from different of its threads instead of maintaining subgroups within the thread and a stack.

In recent GPU architectures, all threads in a warp access the same partition in the SM's register file and communicate through it with low-latency. Blocks of variable sizes m_i can only be emulated by setting the block size to $m = \max\{m_i\}_i$ and masking out threads in each block. A programming model according to embodiments of the invention is proposed on this issue: First, it is gotten rid of the block abstraction and warps are directly exposed to users. In order to handle data irregularity, the individual warps' sizes are made a runtime parameter. Instead of passing parameters m, n to the kernel, this requires passing an explicit list of warp sizes:

```
const int w_list[] = {4, 11, 3, 2, 8, 3};
kernel<<<m, w_list>>>(...);
```

As on GPUs, warps are assigned statically to SMs (SIMT-to-SIMD: vector cores) as illustrated by the example of Fig. 2. Fig. 2 shows a schematic view illustrating an example of distributing warps to SIMD vector cores in accordance with embodiments of the invention. The regular CUDA programming model is extended to irregular execution by having the user input a list of warp sizes, wherein the corresponding (independent) warps are then distributed by a round-robin method to vector cores.

Implementing kernels using CUDA-C follows the same principle as for warp-centric models: all threads in a warp execute the code in a bulk-synchronous manner and threads share data and communicate through *shuffle* instructions. To distinguish code for a model according to an embodiment of the invention from traditional CUDA code, keywords for a thread's index (such as `tid`) in a warp and for a warp's index (such as `wid`) and size (such as `ntids`) are used.

As a posterchild example, the SpMV-kernel given in Fig. 1 (left) is used, in which each warp handles one row of a sparse CSR matrix (arrays `csr_row`, `csr_col`, `csr_val`). Therein, each thread handles one nonzero entry in the row and the results are accumulated by a warp-wide logarithmic reduction (lines 8 through 12 of C source example on the left side of Fig. 1). This simple kernel exhibits both data and control flow irregularity: First, each warp uses a varying amount of threads and thus the share of the SM's register file depends on a runtime parameter where the classical CUDA execution model requires the register count at compiler time. Second, the number of reduction steps depends on the warp size as well, leading to different execution paths for warps of different sizes. Current GPU schedulers would mandate $m_i \leq 32$, leading to a potential waste of resources.

IMPLEMENTATION OF AN EMBODIMENT

In order to natively support varying warp sizes, embodiments of the invention may take the unconventional step of executing programs in a SIMT-focused programming model on traditional SIMD hardware; specifically, according to embodiments, it may be considered to use wide-SIMD (i.e. vector) hardware. A system in accordance with embodiments of the invention may introduce additions to CUDA's C-to-PTX compiler and modifications to vector instruction buffer and

register renaming units in hardware. Both components are frequently found in SIMD microarchitectures (e.g. Intel CPUs with AVX). A fundamental idea of embodiments in accordance with the invention is to translate SIMT code (with SIMD-friendly additions) into SIMD code in hardware and use register naming tables to batch the execution of multiple warps of varying size together.

A. Front-End:

Executing SIMT code efficiently requires hardware support for predicated execution and branch – as well as reconvergence handling (e.g. through stacks per thread). In SIMT models, each thread inside a warp executes the same (scalar) code, but is parameterized by its index inside the warp (CUDA: `lane_id`). SIMD code, on the other hand, has no data or stack per lane, and can only operate on whole vectors. Thus, embodiments in accordance with the invention propose several additions to CUDA's virtual PTX code in order to make it more SIMD-friendly, simplifying processing in the back-end. Such changes are visualized using the SpMV example in Fig. 1.

Metadata registers: Embodiments of the invention may follow the SIMT-on-SIMD paradigm of ISPC (cf. the non-patent literature of M. Pharr and W. R. Mark. "ispc: A SPMD compiler for high-performance CPU programming", In Inpar'12) by mapping the corresponding registers of threads in a warp to lanes in SIMD registers. Rather than at compile time, that mapping is performed on runtime. In this regard, it is noted that the SIMT execution is emulated by explicitly associating keywords such as `tid` in metadata registers to each lane (parameterized SIMD execution). Appearances of those keywords in the code are then translated to registers with suitable execution, as marked by (1) in Fig. 1. Unless there is branching involved, SIMT code translates 1:1 into SIMD code; predicated instruction are realized through masks over SIMD registers. Furthermore, for architectures that support setting a vector length (`vl`) at runtime, a (scalar) metadata register is added.

Scalar branch control: Due to the lack of per-thread program counters (PCs), SIMD hardware is unable to track execution paths of different threads. Instead, all threads must follow the same path of execution. Loops in high-level languages,

e.g., `for`-loops in C, must be repeated until all threads inside a warp evaluate the loop's condition to `false`. Inactive threads' lanes, i.e., threads that evaluate the loop's condition to `false`, are masked out. In order to implement this behavior in accordance with embodiments of the invention, during execution of the program, the loop's condition is then applied as a predicate to all instructions that can potentially follow. Alternatively, if the hardware lacks support for automatic predication, the compiler can combine this predicate (e.g., `%inLoop` as indicated by (3) of Fig. 1) via `and` with all other predicates within the loop to emulate this behavior. Second, a loop is required to be repeated unless its condition evaluates to `false` for all threads in a warp – thus, embodiments of the invention propose warp-synchronous branching commands that only branch if all threads in the warp agree on a predicate's result (as indicated by (2) in Fig. 1 as "`allbra`"). In the following, it is referred to PTX (Parallel Thread Execution) code with these two additions as vector-ready PTX (vrPTX).

The execution resulting in the compiler's transformations as described above, is, in part, specified for a warp of size 4 (`ntids` = 4) in the third column of Fig. 1. Corresponding to line 8 in the C code and (2) in the vrPTX, the predicate `%loop` is computed by comparing `%s` and `$ntids` ("mask build" where "mask" and "predicate" are interchangeable). Through a `popcount`, it is counted how many threads evaluated `%loop` to true. Following the concepts from above, until all threads evaluate the predicate to `false`, the branch in the `allbra` instruction is not taken. This ensures repeated execution of the `for`-loop. When all threads evaluate the predicate to `false`, the program counter is set to `L2`, continuing the execution of all threads after the loop.

B. Back-End:

Thanks to parameterized SIMD execution, differently-sized warps all execute the same vrPTX code. Nevertheless, executing each warp on its own SIMD core would often underutilize the hardware and may prevent from hiding latencies, one of the bedrocks of TOC. As a relief, embodiments of the invention propose modifications to hardware in SIMD systems as follows:

Partitioning: Wide-SIMD systems such as NEC's SX-Aurora processor may offer 16,384 bit wide registers. Following execution models, executing a single warp of size less than 512 would leave many SIMD lanes unoccupied. As noted above, SIMT-like execution can be achieved on SIMD registers by providing the appropriate metadata. According to embodiments of the invention, this fact can be used to pack multiple warps and the warps' data into the same SIMD registers, increasing the vector length `vl` as necessary, wherein to the packed warps is referred as one partition. After warp-wide branching instructions are extended to the whole partition, each instruction can be automatically applied to all warps in a partition; however, the partition's runtime is bound by its longest-running warp due to scalar branch control. In order to avoid heavy partition divergence, an embodiment of the invention propose the following: in the compile phase, the vrPTX source is unrolled for multiple values of `ntids` and the resulting number of instructions is counted. The possible values of `ntids` may be then grouped into buckets according to the difference in their number of instructions; warps that fall into the same bucket may be put into the same partition. According to embodiments of the invention, it may be provided that each partition only requires one program counter (PC) and one slot in the instruction fetch unit.

Vector code issue and multiplexing: While partitioning can be advantageous with warps of similar sizes, there may be still need for an approach of handling warps of drastically different sizes. Even though many SIMD cores offer simultaneous multithreading (SMT), wrapping warps into SMT threads is not an option: with larger SIMD registers, SMT context switches become prohibitively expensive. Instead, according to embodiments of the invention, a static partition multiplex scheme is provided that uses a register rename unit to execute multiple partitions at once. This approach is illustrated by Fig. 3, which shows a schematic view illustrating a register renaming process in accordance with embodiments of the invention. Inside vector cores, a register renaming unit is used to multiplex instruction streams from multiple warps into one vector instruction buffer. Thus, the register renaming unit is employed to multiplex vrPTX instruction streams from multiple warps into one single stream of vector instructions.

As long as vector lengthl `vl` is less than the number of SIMD lanes, all partitions require the same number of SIMD registers. Hence, the process may be implemented analogous to SIMT processors and the register file is divided according to the partitions. In the embodiment of Fig. 3, partition 0 (packing warps 0 and 2) uses physical SIMD registers `v0` through `v4`, partition 1 with warp 4 uses physical SIMD registers `v5` through `v9`. Using the partitioned register table (PRT), incoming vrPTX instructions can be mapped to conflict-free physical SIMD registers. After the mapping, a lookup table performs 1:1 translation from vrPTX to SIDM vector instructions and sets the runtime `vl` accordingly. After renaming, the resulting vector instructions are collected in a buffer. Since there are no dependencies between instruction streams from different partitions, the ability of many SIMD systems may be used to process instructions out-of-order (OoO) to hide latencies by picking other partitions' instructions from the buffer. At this point, all vector instructions are treated equally, without any information about the partition they originated in. Whenever a partition finishes its execution, associated SIMD registers are put back into the PRT's free list; next, the warp dispatcher greedily reads from its assigned warps (as indicated by Fig. 2), builds partitions as necessary and continues execution.

According to embodiments of the invention, SMT capabilities can be used to build virtual cores that map to the same physical core in order to hide latencies between sets of partitions. Hence, all partitions on a core can benefit from the shared instruction cache depending on their PCs and the kernel code length. Furthermore, every SIMD-capable hardware that includes both an OoO instruction buffer and a register may be eligible for a design in accordance with embodiments of the invention.

C. Integration into SX-Aurora:

As a practical example, the modification of a vector processor design that is already on the market is considered: NEC's SX-Aurora TSUBASA.

Fig. 4 shows the architecture of SX-Aurora's VPU. Up to 1 vector instruction is issued by the SPU in each cycle, going into the vector instruction buffer. Out-of-order execution is enabled by a rudimentary register renaming unit that solves

read-after-write and write-after-write conflicts. Instructions are issued into 32 vector processing pipelines (VPPs) with 8 vector registers and 3 execution ports each.

Aurora's models 20x are PCIe cards that offers up to 10 cores running at 1.6 GHz, achieving up to 3.07 TFLOPs in double precision mode. All cores share 16 MB last-level cache (LLC), interfacing with 48 GB of HBM2 memory with a sustained bandwith of 1.53 TB/s. Each Aurora core features a relatively simple, out-of-order scalar processor (SPU) that can issue one instruction per cycle to the core's vector processing unit (VPU) as illustrated by Fig. 4. Each VPU uses register renaming to resolve WAW and WAR dependencies and reorders vector instructions before issuing them to the 32 vector processing pipelines (VPP) per core. VPPs, in turn, each include 8 vector registers with 16,384 bit, 2 mask registers and 3 execution ports to FMA units and ALUs (only 64 architectural vector registers are exposed through the ISA).

For a design in accordance with an embodiment of the invention, it is focused on the VPU and the instruction fetching capabilities of the SPU are used. Fig. 5 shows a schematic view illustrating an architecture overview of a vector core for a system in accordance with embodiments of the invention. A scheme for warp multiplexing by register renaming in accordance with an embodiment of the invention is integrated into SX-Aurora's vector core. Due to the focus on the offload mode, the SPU is removed except for its instruction fetch and decode units and the conventional register renaming unit is replaced by an embodiment as illustrated by Fig. 3 (cf. dotted box in Fig. 5).

Fig. 5 depicts modifications with regard to the architecture of Fig. 4 as follows: compared with the original VPU architecture (see Fig. 4), the register renaming unit is pulled before the vector instruction buffer (as illustrated by Fig. 5), since the OoO-execution is not used within partitions. Instead, vrPTX instructions are loaded for up to 4 partitions (using the SPU's 4-way instruction fetch) and multiplexed into a single vector instruction stream to the vector instruction buffer. The remainder (and majority) of the architecture can be maintained unchanged.

Thus, embodiments of the invention have the potential of improving the execution of irregular code for various throughput-oriented architectures. By implementing incremental changes that leave most of the existing architectures' overall designs intact, a comparatively cheap way can be offered to leverage existing systems for efficient irregular processing.

Fig. 6 shows a schematic view illustrating a method in accordance with an embodiment of the present invention. The embodiment outlined by Fig. 6 executes scalable SIMT-style code with different runtime configurations on SIMD processors using metadata registers that are programmed at runtime (cf. Fig. 6 (a)). Furthermore, several warps are dynamically packed with different configurations into the same SIMD registers such that they can be executed simultaneously through metadata concatenation (cf. Fig. 6 (b)). Both approaches are used to achieve irregular SIMT execution on a (wide-) SIMD processor.

Fig. 7 shows a schematic view illustrating a process of metadata register programming for processing warps and partitioning of multiple warps into a single partition.

SIMULATION RESULTS OF AN EMBODIMENT

An embodiment of the present invention extends CUDA's familiar programming model and implement SIMT-inspired strategies for dealing with data and control flow irregularities. The approach requires only minimal hardware changes and an additional compiler phase. It could be demonstrated using a model-based software simulation that the proposed system can be a step towards native support for irregularity on throughput-oriented processors while greatly simplifying the development of irregular applications.

Figs. 8, 9 and 10 shows results from a model-driven simulation of a setup in accordance with an embodiment of the invention, the setup running an SpMV kernel for two sparse matrices (upper row: lp22, lower row: mycielskian11) with one order of magnitude irregularity. The results show that the architecture modifications in accordance with an embodiment of the invention help to make efficient use of the execution units and hide latencies.

Due to a lack of details regarding NEC's SX-Aurora, the available ISA documentation was used in order to build a model following Fig. 4. Then it was simulated using the SimPy framework (cf. K. G. Müller and T; Vignaux; Simpy; retrievable at https://github.com/cristiklein/simpy). All latencies are expressed in terms of multiples of a simple arithmetic vector operation that takes 1 cycle, any complex resp. store operation's latency is the active vector length. One core of the vector processor was simulated with the same number of memory controllers and ports as Aurora. The simulation consumes the generated PTX from the SPMV example code (as illustrated by Fig. 1). Multiple matrices are inputted from the SuiteSparse Matrix Collection's linear programming category (cf. non-patent literature of T. A. Davis and Y. Hu.: "The University of Florida sparse matrix Collection", ACM TOMS, 38:1–25, 2011), since these matrices often suffer from data irregularity (i.e. different row lengths). This test is meant to showcase two things: First, allowing more partitions (t – also the number of required instruction fetch units) per core results in a larger vector instruction buffer which leads to better utilization of the execution units and thus less empty cycles. Second, packing can save instructions by batching warps – again, it is expected less time to termination.

Figs. 8-10 presents simulation results for two matrices that are representative for the test set: lp22 (2, 958×16, 392; 68, 512 nz – first row) and mycielskian11 (1, 535 × 1, 535; 134, 710 nz – second row). Their row length distributions, and thus warp size distributions, are visualized in Fig. 8. Fig. 9 supports a first hypothesis: Independent from the packing setup, more slots result in consistently less cycles being used. More slots lead to more and potentially different simultaneous instructions in the instruction buffer which in turn may be executed in parallel (pending execution unit availability). Furthermore, a looser threshold for packing (permitting higher warp size variations inside a partition) further reduces the total number of cycles spent. In Fig. 10, the execution unit utilization is visualized in the same experiments: Again, both more partitions as well as looser packing thresholds increase the utilization until reaching a plateau. Lastly, it is pointed to the error bars for t = 4 (Figs. 9, 10): for each parameter setting, the simulation was run 100 times, every time with a random order of the input matrix' rows, and the

resulting error bars in both cycle and utilization plot are plotted. It is pointed out that although the variation is relatively large, at times negating the benefit of packing entirely, the average line (black) tends strongly towards the better region (lower cycles, higher utilization). This indicates that a smaller number of outliers is responsible for such failures. Since there is currently no support for work stealing or dynamic allocation, these outliers directly correspond to certain row orders.

Many modifications and other embodiments of the invention set forth herein will come to mind to the one skilled in the art to which the invention pertains having the benefit of the teachings presented in the foregoing description and the associated drawings. Therefore, it is to be understood that the invention is not to be limited to the specific embodiments disclosed and that modifications and other embodiments are intended to be included within the scope of the appended claims. Although specific terms are employed herein, they are used in a generic and descriptive sense only and not for purposes of limitation.

Claims

1. A method for supporting throughput-oriented computing, in particular in a high-performance computing system,

wherein a single instruction multiple threads, SIMT, program is configured to launch warps,

wherein each warp comprises threads to be executed in lockstep within their warp,

wherein the warps' individual warp sizes are used as a runtime parameter for the SIMT program, such that a parameterized SIMT program is provided, which is parameterizable via the warp sizes, and

wherein the parameterized SIMT program is executed on a single instruction multiple data, SIMD, vector architecture.

2. The method according to claim 1, wherein a programming model is provided, which is based on a SIMT programming model and which is extended for handling data irregularity by having a user input a list of warp sizes.

3. The method according to claim 1 or 2, wherein the warps are distributed, in particular by using a round-robin method, to SIMD vector cores of the SIMD vector architecture.

4. The method according to any one of claims 1 to 3, wherein the SIMT program is mapped onto SIMD vector registers of the SIMD vector architecture by using predetermined vector registers that are initialized by SIMT metadata of SIMT metadata registers.

5. The method according to claim 4, wherein the SIMT metadata includes a thread index information, a warp index information and/or a warp size information.

6. The method according to any one of claims 1 to 5, wherein a mapping of the threads' SIMT metadata registers to lanes in SIMD vector registers of the SIMD vector architecture is performed at runtime.

7. The method according to any one of claims 1 to 6, wherein the SIMT program is compiled to an intermediate code representation, wherein said intermediate code representation is configured to replace one or more identifiers pertaining to the SIMT program's runtime configuration with references to SIMT metadata registers.

8. The method according to claim 7, wherein the intermediate code representation includes a scalar branch control such that warp-synchronous branching commands are provided that only branch if all threads in the warp agree on a predicate's result.

9. The method according to claim 8, wherein a predicate is created whenever branching is involved in the SIMT program, wherein the predicate indicates whether a thread took a branch or not, and wherein the predicate's result is applied to all instructions that can follow.

10. The method according to any one of claims 1 to 9, wherein a control flow of the SIMT program is modified to emulate branching by using a stack of masks for SIMD vector registers, wherein a translation from thread-wide branching instructions to warp-wide branching instructions is provided.

11. The method according to any one of claims 1 to 10, wherein a partitioning scheme is implemented for executing partitions on a SIMD vector core of the SIMD vector architecture, wherein a partition includes one or more warps to be executed on the SIMD vector core.

12. The method according to claim 11, wherein several warps included in a partition are packed into a SIMD vector register by concatenating their metadata, and wherein a vector length of the SIMD vector register is increased accordingly.

13. The method according to any one of claims 1 to 12, wherein a register renaming is performed inside SIMD vector cores in order to multiplex instruction streams from multiple partitions into a vector instruction buffer.

14. The method according to claim 13, wherein the register renaming is performed based on a partitioned register table.

15. A system for supporting throughput-oriented computing, in particular for execution of a method according to any one of claims 1 to 14, the system comprising a programming model and a SIMD vector architecture,

wherein the programming model is configured

to provide a SIMT program for launching warps, wherein each warp comprises threads to be executed in lockstep within their warp, and

to use the warps' individual warp sizes as a runtime parameter for the SIMT program, such that a parameterized SIMT program is provided, which is parameterizable via the warp sizes, and

wherein the SIMD vector architecture is configured to execute the parameterized SIMT program.

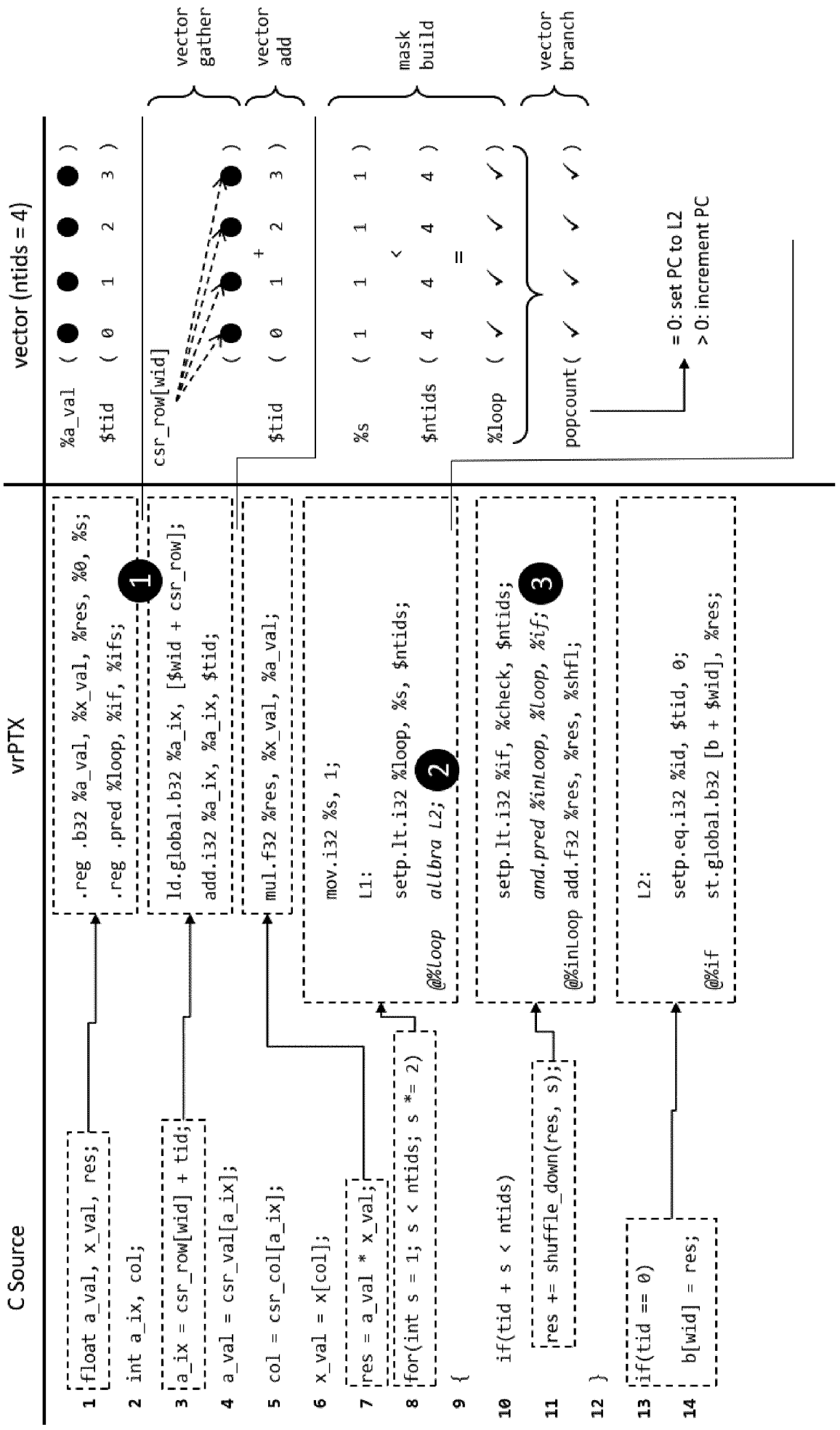

C Source
vrPTX
vector (ntids = 4)
vector gather
vector add
mask build
vector branch
1
2
3
= 0: set PC to L2
> 0: increment PC

Fig. 1

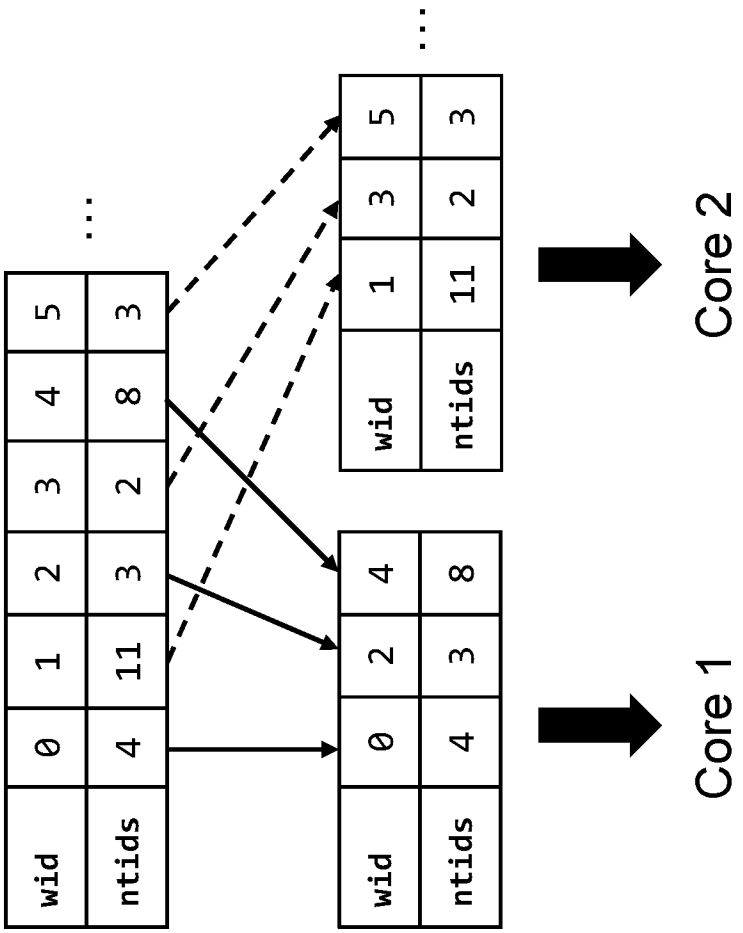

wid
ntids
0
4
1
11
2
3
3
2
4
8
5
3
...
wid
ntids
0
4
2
3
4
8
1
11
3
2
5
3
...
Core 1
Core 2

Fig. 2

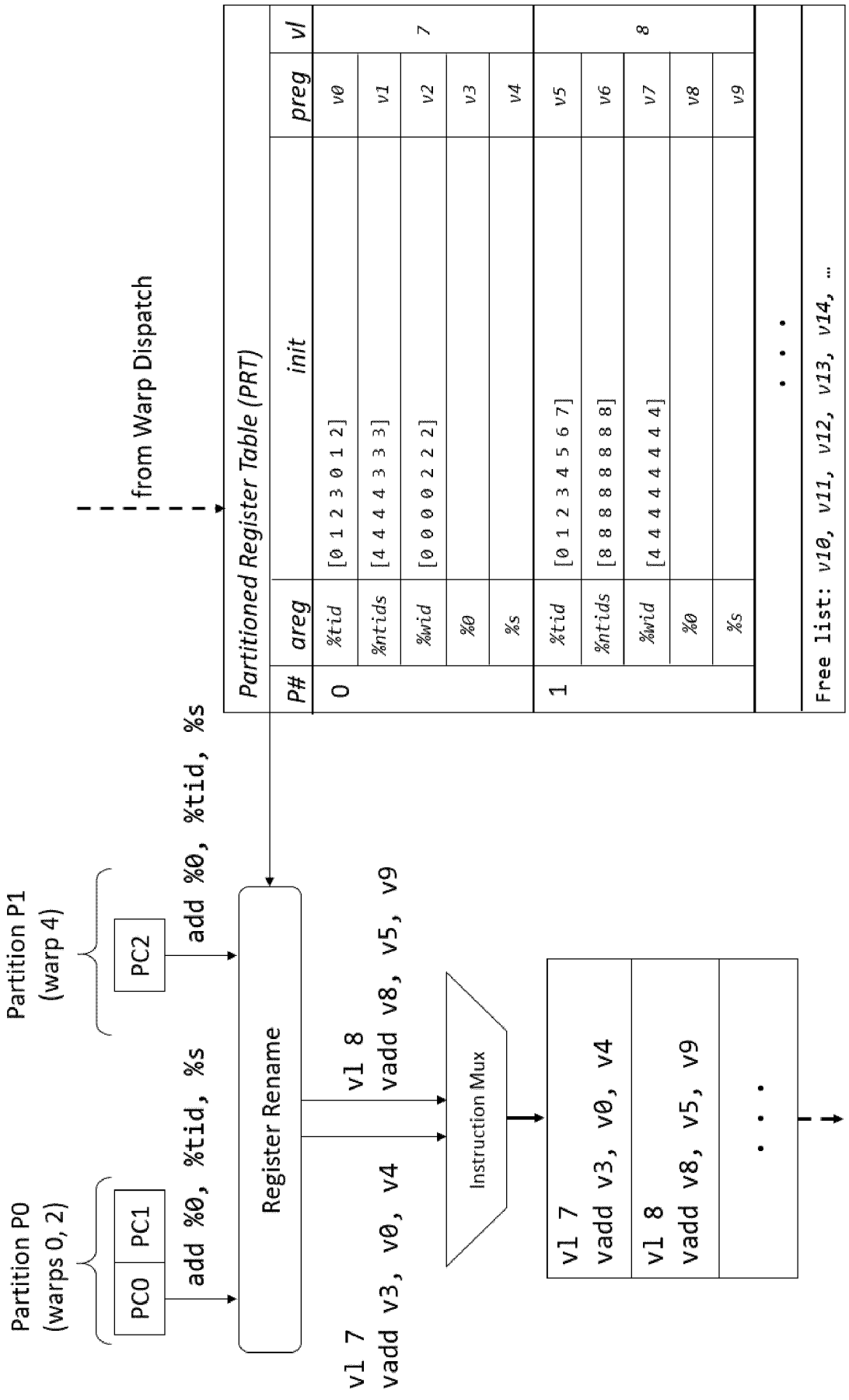

Partition P0
(warps 0, 2)
PC0
PC1
Partition P1
(warp 4)
PC2
add %0, %tid, %s
add %0, %tid, %s
Register Rename
v1 7
vadd v3, v0, v4
v1 8
vadd v8, v5, v9
Instruction Mux
v1 7
vadd v3, v0, v4
v1 8
vadd v8, v5, v9
. . .
from Warp Dispatch
Partitioned Register Table (PRT)
P#
areg
init
preg
vl
0
%tid
[0 1 2 3 0 1 2]
v0
%ntids
[4 4 4 4 3 3 3]
v1
%wid
[0 0 0 2 2 2]
v2
7
%0
v3
%s
v4
1
%tid
[0 1 2 3 4 5 6 7]
v5
%ntids
[8 8 8 8 8 8 8 8]
v6
%wid
[4 4 4 4 4 4 4 4]
v7
8
%0
v8
%s
v9
. . .
Free list: v10, v11, v12, v13, v14, …

Fig. 3

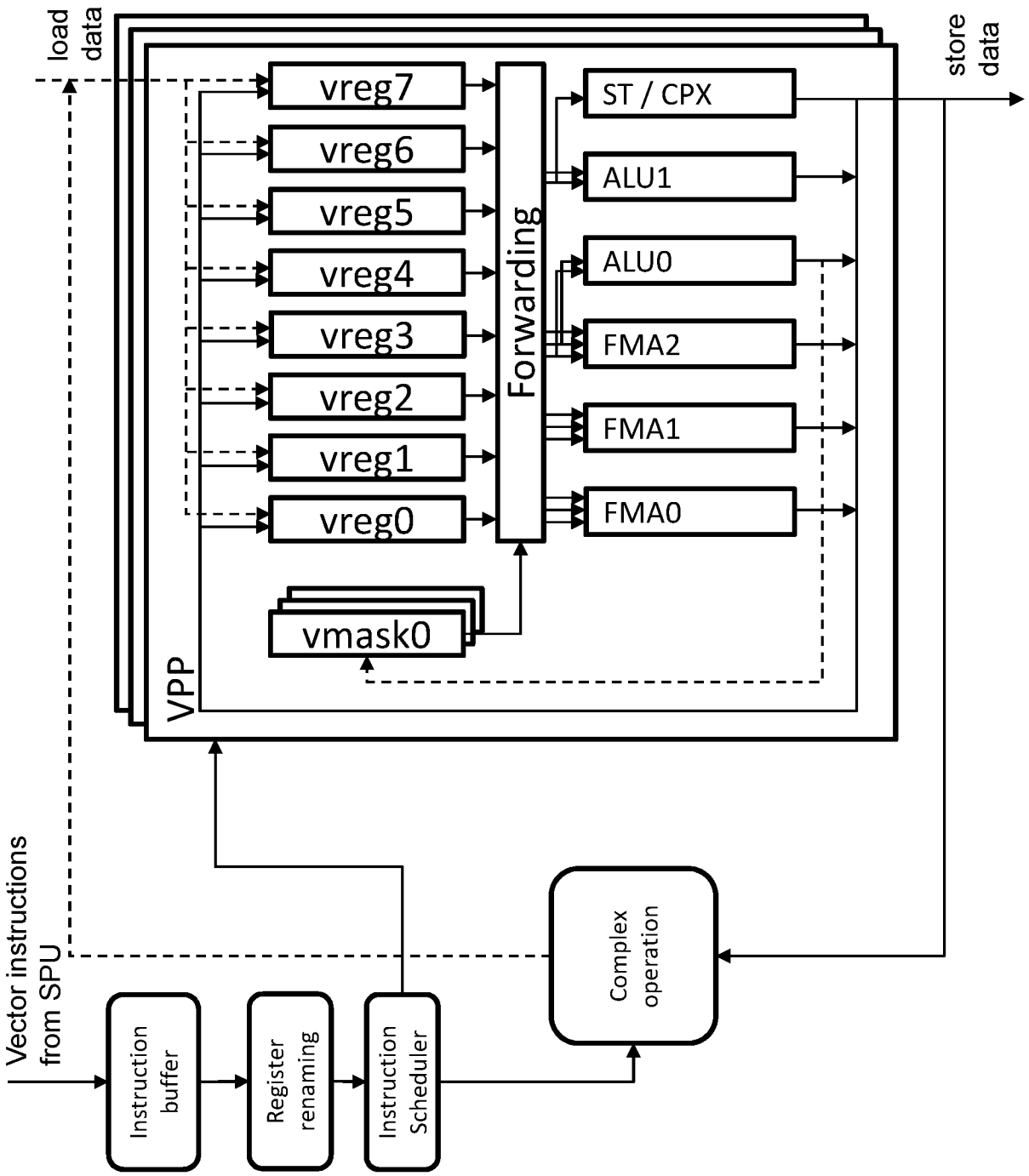
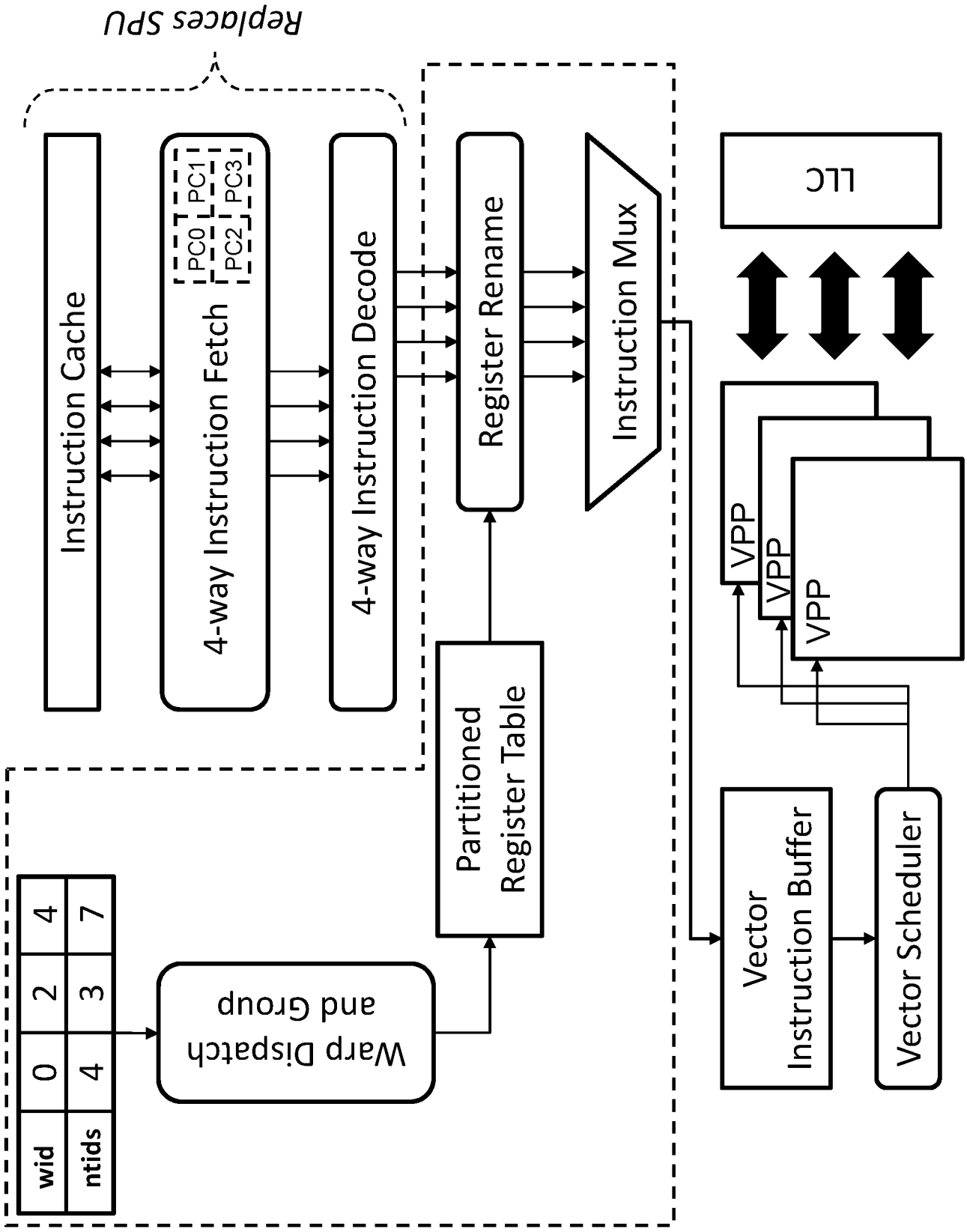

load data
store data
vreg7
vreg6
vreg5
vreg4
vreg3
vreg2
vreg1
vreg0
Forwarding
ST / CPX
ALU1
ALU0
FMA2
FMA1
FMA0
vmask0
VPP
Vector instructions from SPU
Instruction buffer
Register renaming
Instruction Scheduler
Complex operation

Fig. 4

Fig. 5


Replaces SPU
Instruction Cache
4-way Instruction Fetch
PC0
PC1
PC2
PC3
4-way Instruction Decode
Register Rename
Instruction Mux
LLC
VPP
VPP
VPP
Partitioned Register Table
Warp Dispatch and Group
wid
ntids
0
4
2
3
4
7
Vector Instruction Buffer
Vector Scheduler

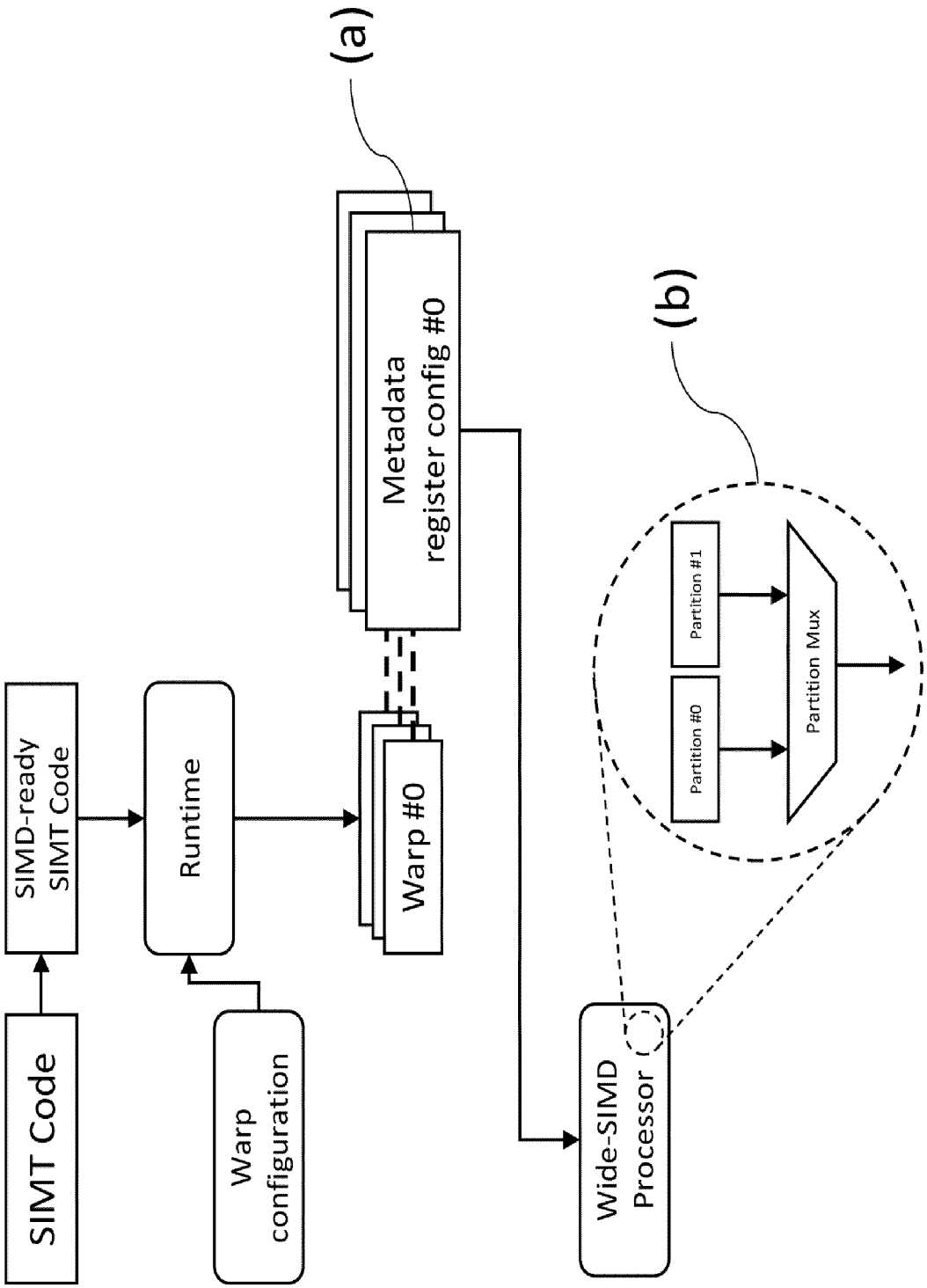

(a)
Metadata register config #0
(b)
Partition #1
Partition #0
Partition Mux
SIMD-ready SIMT Code
Runtime
Warp #0
SIMT Code
Warp configuration
Wide-SIMD Processor

Fig. 6

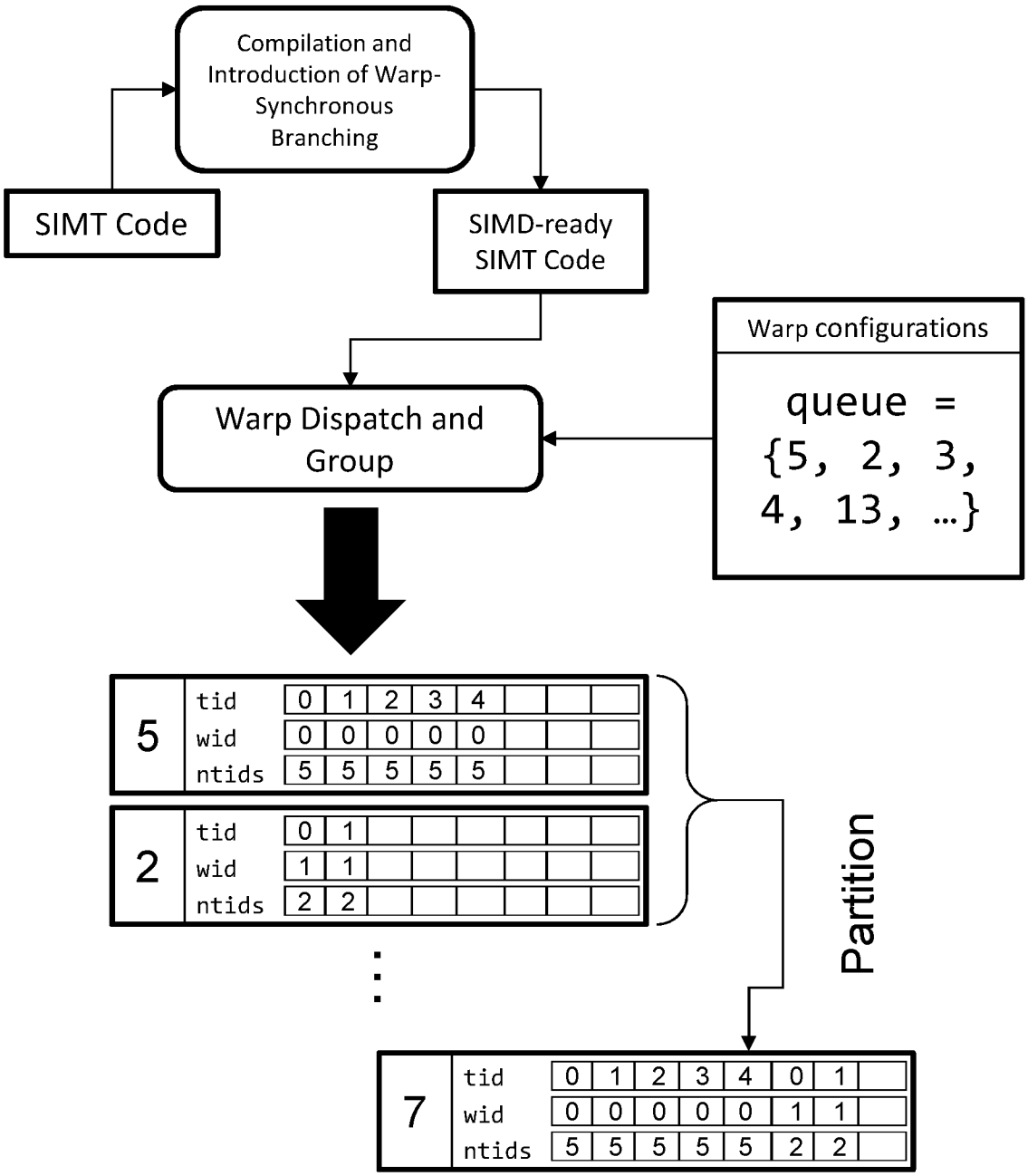

Compilation and Introduction of Warp-Synchronous Branching
SIMT Code
SIMD-ready SIMT Code
Warp configurations
queue = {5, 2, 3, 4, 13, …}
Warp Dispatch and Group
5
tid 0 1 2 3 4
wid 0 0 0 0 0
ntids 5 5 5 5 5
2
tid 0 1
wid 1 1
ntids 2 2
Partition
7
tid 0 1 2 3 4 0 1
wid 0 0 0 0 0 1 1
ntids 5 5 5 5 5 2 2

Fig. 7

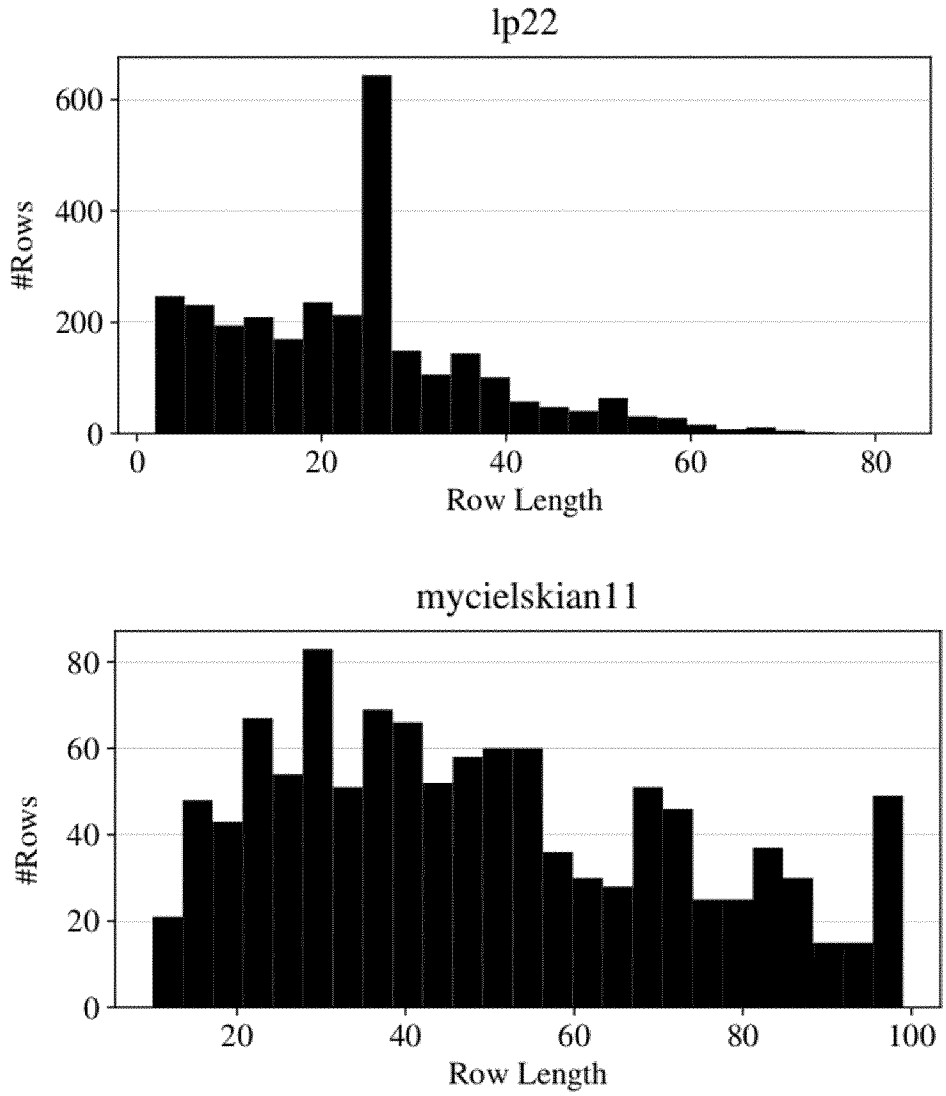

lp22
#Rows
0
200
400
600
0
20
40
60
80
Row Length
mycielskian11
#Rows
0
20
40
60
80
20
40
60
80
100
Row Length

Fig. 8

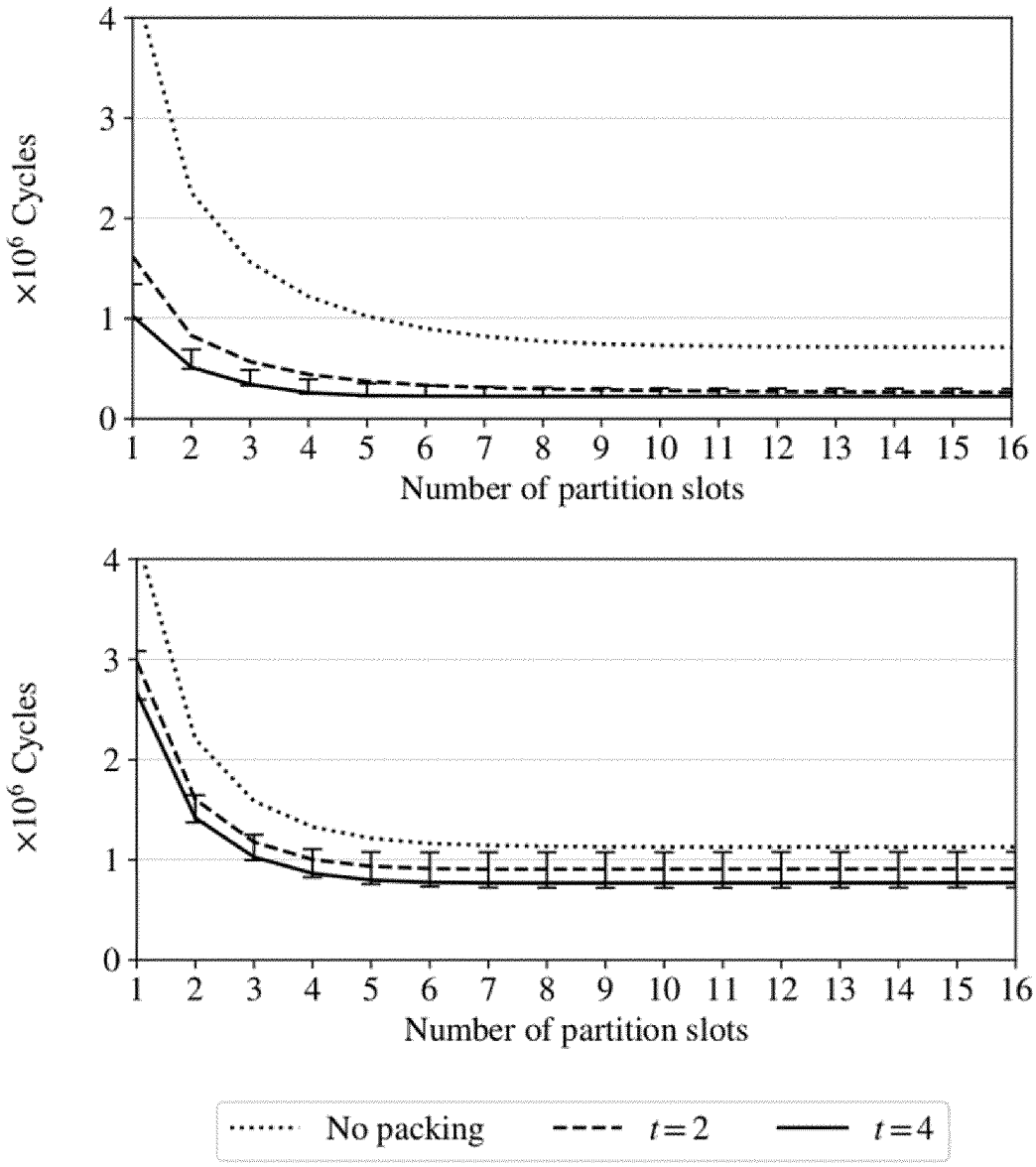

×10^6 Cycles
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Number of partition slots
×10^6 Cycles
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Number of partition slots
No packing
t=2
t=4

Fig. 9

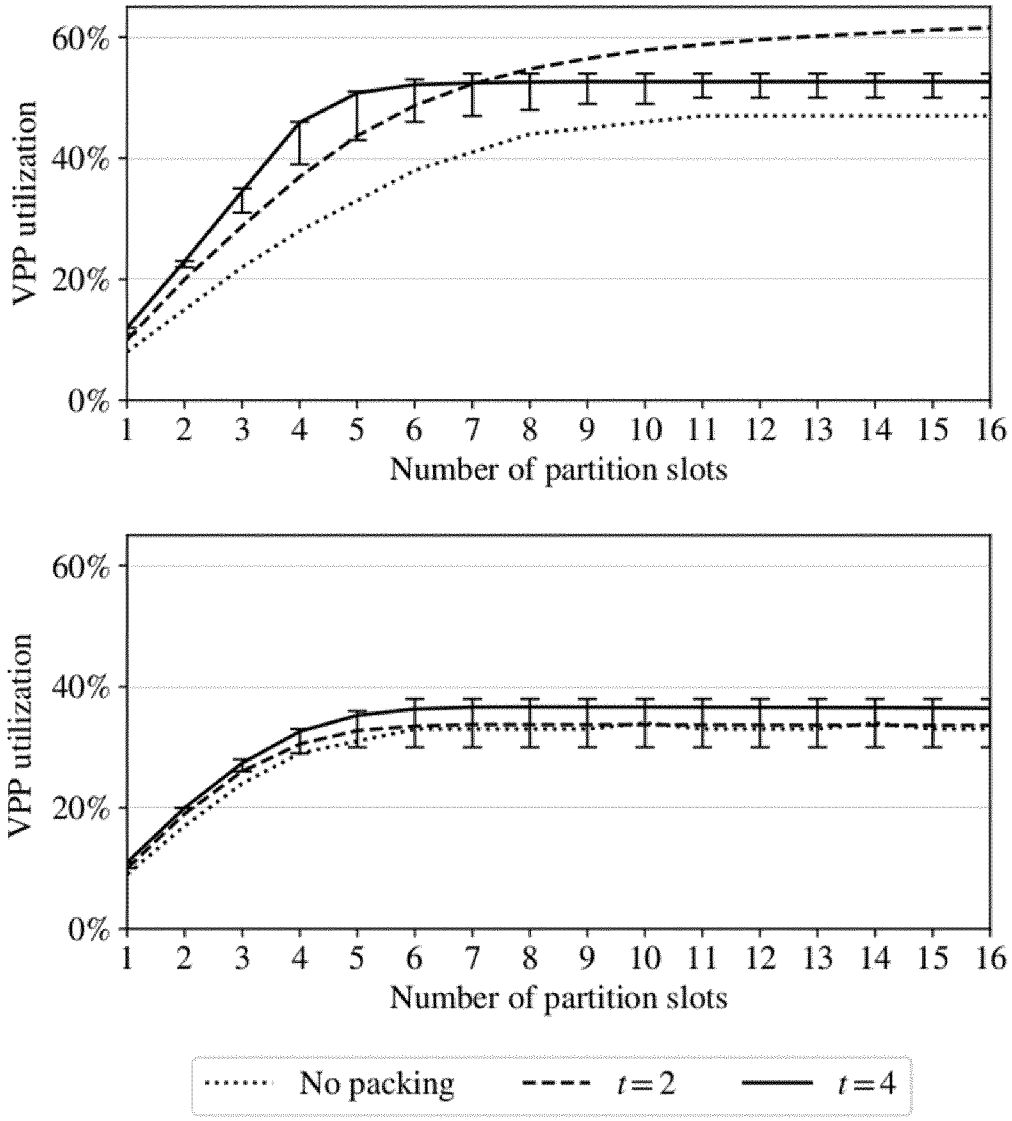

60%
40%
20%
0%
VPP utilization
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Number of partition slots
60%
40%
20%
0%
VPP utilization
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Number of partition slots
No packing
t=2
t=4

Fig. 10

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2021/054748

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/30 G06F9/38
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Lashgar Ahmad ET AL: "Dynamic Warp Resizing in High-Performance SIMT", ICCD 2012, 11 August 2012 (2012-08-11), pages 1-9, XP055810945, Retrieved from the Internet: URL:https://arxiv.org/ftp/arxiv/papers/1208/1208.2374.pdf [retrieved on 2021-06-07]	1-7,14, 15
Y	abstract; figures 1-5 page 2, left-hand column - page 9, right-hand column	8-13

-/--

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 June 2021	23/06/2021

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Freitas, Arthur

2

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	FUNG WILSON W L ET AL: "Energy efficient GPU transactional memory via space-time optimizations", 2013 46TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE (MICRO), ACM, 7 December 2013 (2013-12-07), pages 408-420, XP033061514, DOI: 10.1145/2540708.2540743 [retrieved on 2017-02-08]	11-13
A	abstract; figures 1-16 page 410, left-hand column - page 420, left-hand column	1-10,14, 15

A	YEH TSUNG TAI YEH14@PURDUE EDU ET AL: "Dimensionality-Aware Redundant SIMT Instruction Elimination", PROCEEDINGS OF THE TWENTY-FIFTH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, ACM, NEW YORK, NY, USA, 9 March 2020 (2020-03-09), pages 1327-1340, XP058460503, DOI: 10.1145/3373376.3378520 ISBN: 978-1-4503-7102-5 abstract; figures 1-12 page 1330, left-hand column - page 1337, left-hand column	1-15

Y	MAYANK KOTHIYA ET AL: "Analyzing graphics processor unit (GPU) instruction set architectures", 2015 IEEE INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS), IEEE, 29 March 2015 (2015-03-29), pages 155-156, XP032771511, DOI: 10.1109/ISPASS.2015.7095794 [retrieved on 2015-04-27]	8-10
A	abstract; figures 1-3b page 155, left-hand column - page 156, right-hand column	1-7, 11-15

Y	BRUNO COUTINHO ET AL: "Performance Debugging of GPGPU Applications with the Divergence Map", COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING (SBAC-PAD), 2010 22ND INTERNATIONAL SYMPOSIUM ON, IEEE, PISCATAWAY, NJ, USA, 27 October 2010 (2010-10-27), pages 33-40, XP031807844, ISBN: 978-1-4244-8287-0	8-10
A	abstract; figures 1-5 page 35, left-hand column - page 39, right-hand column	1-7, 11-15
