Validating Timing And Scheduling Marte's Profils Using Event B: Case Study Of A Gpu Architecture

Imane ZOUANEB, Mostefa BELARBI, Chouarfia Abdellah
LIM Research Laboratory, University of Ibn khaldoun Tiaret,Algeria
i_zouaneb@yahoo.fr, belarbimostefa@yahoo.fr

Abstract-*System on chip multi calculator (CPU and GPU) is a promoted filed to parallelize application thanks to the multi-core GPU architecture. GPUs (Graphic Processing Unit) ensure the parallelism on the chip and discharge the Central Processing Unit (CPU). The specification of scheduling and timing on GPUs had been always a research problematic. MARTE is an efficient semi formal tool for specification thanks to the several diagrams of UML and the new profiles provided by MARTE which treats the software, hardware and scheduling of the specified SoC. But it still none valid specification because it isn't proved. That's why we propose to couple MARTE with the formal method Event B to have a valid and proved specification and to validate the task scheduling on the GPU. After having a valid specification a second phase of executable code generation from Event B specification is essential to execute parallel applications on the GPU. CUDA is an efficient programming language on GPUs because it offers new tools for parallel programming.*

Index Terms

GPU, MARTE, Scheduling, Timing Event B, Refinements, Code generation, CUDA.

1. INTRODUCTION

A System on Chip (SoC) is a total electronic system integrated on one chip. A SoC can be constituted of a CPU, a memory (DRAM), a bus and a specialized unit of processing according to the SoC's function. In the last years the Graphic processing Unit (GPU) started to be essential in SoCs. GPUs permit to execute parallel tasks on the SoC.

To specify SoCs we need specialized tools such as UML[10]/ MARTE (Modeling and Analysis of Real-Time and Embedded Systems) which offer a support to cover all the phases of SoC development and to specify hardware and software SoC's aspects. But this specification misses the mathematic improves because it's informal so it can't be considered valid. To solve this problem the proposed solution is to couple the UML/MARTE with a formal tool to have a sure specification based on mathematic notions and successive refinements.

We are interested to coupling UML/MARTE to the formal method B-event which is an extension of B method. Many works have proposed approaches to translate UML diagrams to B specification. The work made by Laleau [1] proposes a tool of automatic generation of class and state-transition diagram to B abstract machines Using OCaml language in Rose Programming Environment. But he found some limits of semantics of the concepts which cause the user intervention to complete the generated specification. Then Ledang [2] proposed an approach to translate the comportment diagrams because according to him the previous works have interested just to static diagrams. He has concentrated on collaboration diagram by considering it as layers of objects. He proposed the notion of calling-called to link between layers and generated abstract machines. We can say that Ledang [2] has created an effective tool to translate UML comportment diagrams to B specification. In addition, to complete his works, Ledang [3] has created a tool named ArgoUML+B which permit to generate B specifications from UML diagrams. This tool has given good results in the field of UML translation to B specification and it has been developed using Java to avoid limits founded in Laleau's work[1]. Based on this works we propose an approach of coupling UML/MARTE to B-event specification to have proved and verified specification thanks to B-event.

After specifying our SoC we need to generate an executable code on GPU from the MARTE specification using the proved Event B specification as an intermediate. Model Driven Engineering (MDE) proposes an approach to generate executable code from a model throw successive transformations of the semi-formal specification. In this field several works have been proposed by

[10]UML: Unified Modeling Language

Wendell [4][5] to generate OpenCL[11] code from a GPU MARTE specification in order to provide a tool of code generation for none specialized in parallel programming to develop their applications. Wendell proposed an MDE approach of specification, modeling and generation of OpenCL applications. A first specification has been done using UML/MARTE and ARRAYOL of GPU architecture and the Conjugate Gradien algorithm, then a successive transformations using MOF/QVT (Meta-Object Facility Query/View/Transformation) result an executable OpenCL code on GPU.[4] Another work which studies a H.263 video compression (Downscaling) application where a preliminary MARTE specification of the downscaler has been done using Gaspard2. Then an OpenCL valid code has been generated from Gaspard2 specification.[5] We are going to generate a pre-code CUDA (Compute Unified Device Language) from UML/MARTE specification but after transforming this later into Event B specification to guarantee its validity.

This paper is divided into several sections, section 2 defines MARTE specification. The section 3 gives a brief description of Event B. In section 4 we describe GPU's architecture. Then we present our proposition of GPU specification in section 5 and we treat a study case of GPU specification using UML/MARTE, coupling MARTE with Event B and extracting a CUDA code of vector addition algorithm using Event B refinements. Finally we present our conclusion and some perspectives.

2. MARTE

The MARTE (Modeling Analysis Real Time Embedded systems) profile is an extension of UML to complete the missing tools of embedded systems modeling. MARTE was created by "ProMarte" consortium for OMG (Object Management Group) users. [6] MARTE is composed of four packages: foundations, design model, analysis model and annexes. Each package contains several profiles which permit specifying, modeling, analyzing and verifying an embedded real time system.

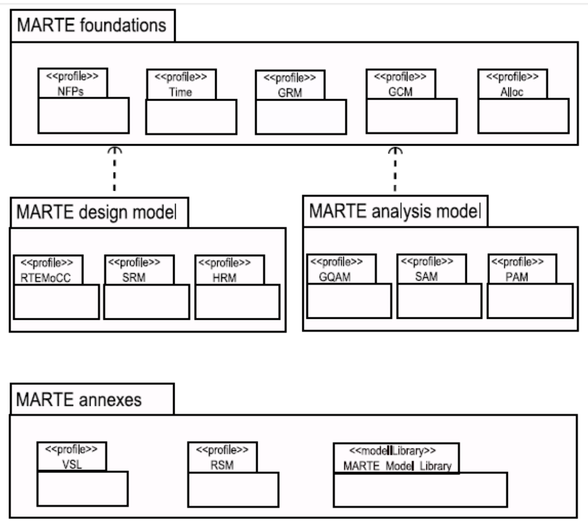


Fig 1. MARTE architecture

MARTE has improved the UML specification because:

- It distinguishes the hardware part from the software part using Hardware Resource Modeling (HRM) profile and Software Resource Modeling (SRM) profile.
- It permits to allocate the software application on the hardware resources thanks to Allocation profile (Alloc).
- It permits timing and scheduling modeling.
- It permits performance and scheduling analysis using Performance Analysis Modeling (PAM) profile and Schedulability Analysis modeling (SAM) profile.

Several works have used MARTE profile for system modeling especially real time embedded system thanks to its efficient tools (profiles). We mention the work of remote controlled robot specification [7] where they used MARTE for real time constraints modeling. They stereotyped the classes with *SchedulableResource* stereotype in class diagram and they added *timing observation (reference)* in sequence diagram to illustrate the timing constraints of remote controlled robot system. Another paper where they dealt with time specification in an automotive system of an ignition control and knock correction in the case of four stroke engine. In a 4-stroke engine a cycle is composed of four phases: Intake, Compression, Combustion and Exhaust. This phases where represented by a timing diagram to illustrate the timing properties in addition to the MARTE notions of *TimedEvent* and *TimedProcessing* used in the state-transition diagram.[8] The MoPCom approach which is a co-design methodology to generate VHDL codes has also used MARTE to describe real-time properties and perform the platform modeling. In addition to UML diagrams, the MARTE profiles SRM, HRM and Alloc have been essential in the process of VHDL code generation.[9] MARTE was also a base of a methodology approach for high level modeling and model+code generation for embedded real time systems. The methodology consist of specifying a system with UML and MARTE profile then the specification become a source to model and code generation of real time components and scheduling analysis. [10] MARTE became an essential element in real time embedded systems specification because it offers a multitude tools (profiles) for modeling time constraints, scheduling and performance of systems.

3. EVENT B

Event B is an enriched extension of the formal method B created by J. R Abrial [11] for system specification, design and coding. It is based on Set theory and it specifies the system by abstract machines, operations and successive refinements which permit to prove, to verify and to validate the specified system.

Event B is based on MODEL notion which describes the labeled transaction of the system. A MODEL is composed of a static part which contains the states, its invariants and its properties and a dynamic part containing transitions (events). A MODEL has a name, variants, invariants and Events. A MODEL is completed by a formalism celled the CONTEXT. It plays an important role in MODEL parameterization and instantiation. A CONTEXT has also a name, Sets, Invariants.[12][13] Each MODEL can reference a CONTEXT and many refinements which concretes models and contexts as it is shown in the figure 2.

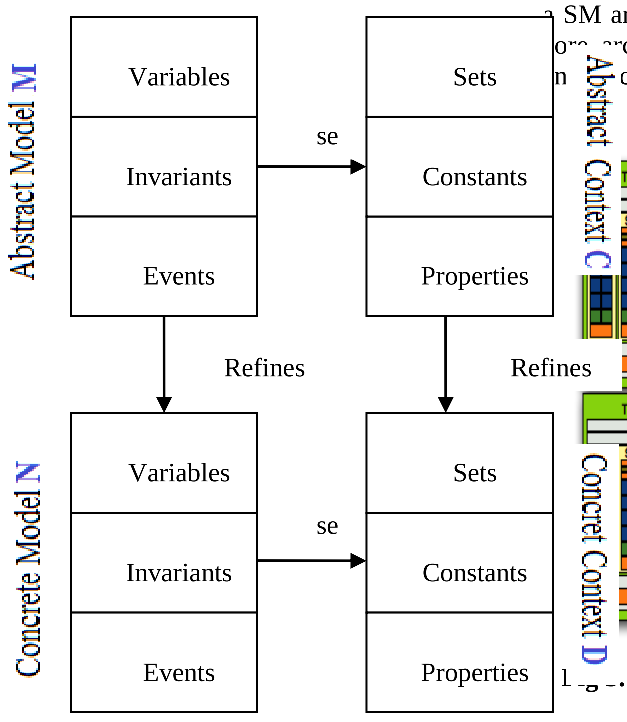


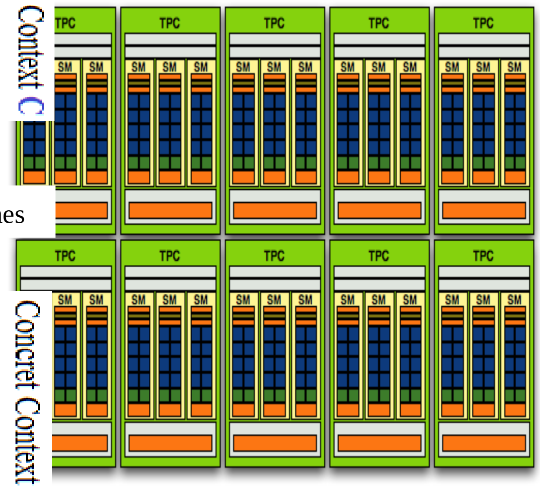
Fig 2. Refinements of models and contexts

The Event B method is efficient because it uses tools like Atelier B[12] and the platform RODIN (Rigorous Open Development Environment for Complex Systems). This platform is a tool to develop and to prove Event B specification under Eclipse environment. [12] The main objective of RODIN is to create a methodology and supporting open tool platform for cost-effective, rigorous development of complex, dependable software systems and services. [14]

4. GPU ARCHITECTURE

Graphic Processing Units have a high performance processors dedicated to graphics processing. Originally, GPUs were oriented to accelerating graphics rendering functionality. Lately they are used to perform different kinds of general purpose computations in a parallel way to minimize application's runtime.[15]

GPU is a multi-core architecture used to enhance intensive computing and to discharge the CPU. A GPU is composed of a Global memory (DRAM) and a set of Streaming Multiprocessor (SM). Each SM is constituted of a set of Streaming Processor (SP) and each SP is linked to a local memory (Register memory). And the SPs of a SM are linked to a shared memory. The multi-core architecture of GPU ensures its efficiency in capacity of computing. [16]



[illegible] GeForce GT GPU architecture

In Nvidia architecture tasks are executed using SIMD (Singel Instruction Multiple data) blocs written in CUDA. [17] CUDA (Compute Unified

[12] Atelier B is a tool that permit operational use of the method B : http://www.atelierb.eu

Device Architecture) provides a set of software libraries, an execution environment and a multitude drivers for different languages of programming (C,C++,…). CUDA is an extension of C language for programming on NVIDIA GPU. The computations on a GPU are programmed as *kernel* functions. A kernel program describes the execution of a serial *thread* on a GPU. The kernel is launched by the host CPU with specified numbers of blocks and threads, where a *block* represents a set of a certain number of threads, and all blocks in that kernel launch have the same numbers of threads. The total number of threads is the number of blocks times the number of threads per block. [18] Since there is a number of processing units on GPU a solution of scheduling is needed to organize the execution on GPU.

5. PROPOSED APPROACH VIA A CASE STUDY

Our main objective is to specify a GPU with different tools and to generate a valid executable code.

We propose to use UML diagrams and MARTE profile to specify a SoC with:

i. Hardware Resource Modeling (HRM) to specify the SoC components,

ii. Software Resource Modeling (SRM) for modeling the applications that will be executed on the SoC.

iii. Allocation (Alloc) profile to allocate the software on the hardware components.

iv. **Timing profile** for time constraints modeling.

v. **Schedulability Analysis Modeling** (SAM) for scheduling modeling and analysis.

After specifying our SoC with MARTE profile we propose to couple it with Event B specification to make it valid, sure and proved with Event B rigorous tools. We have proposed a set of rules for transformation of MARTE models into Event B specification.

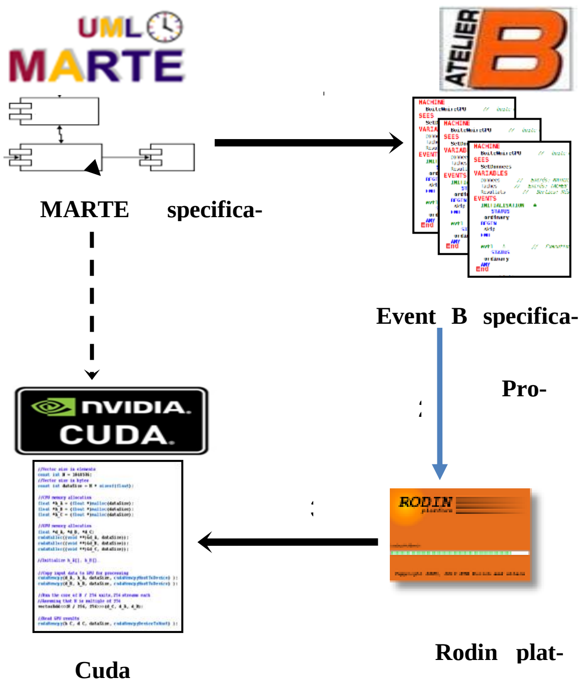


Fig 4. Proposed approach of SOC specification

Once a valid, sure and correct specification of SoC has been carried out, it could be exploited to generate an executable code to be run on the SoC hardware resources (CPU/GPU). We propose to do a set of refinements of Event B specification to generate a parallel executable code written in CUDA language which is rich of parallel implementation on NVIDIA GPU architectures.

To test our approach we used a case study of GPU architecture. We are going to specify a GPU architecture using MARTE and Event B with an application (***Vector addition***) which will be executed on GPU architecture.

5.1. GPU specification

5.1.1 GPU architecture

We are using a GeForce GT 210 GPU to implement our case study. It is composed of 16 CUDA cores. The relation between internal GPU components is illustrated in the component diagram where each component is stereotyped by MARTE stereotypes for each type component (hwcomponent, hwRam, hwBus, hwComputingResource,…).

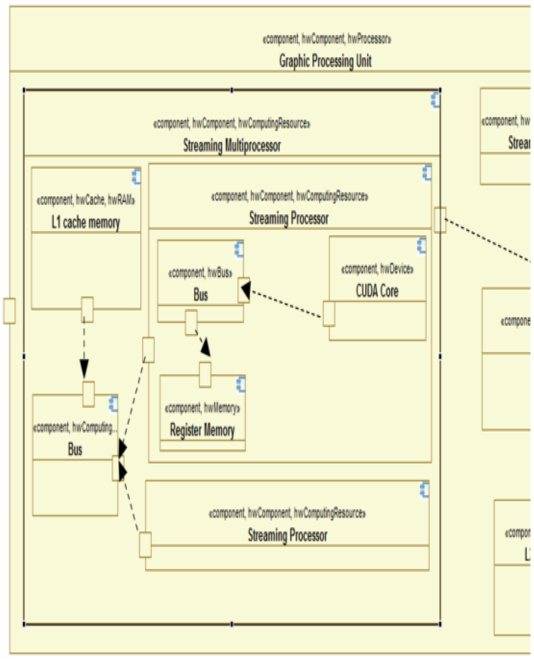
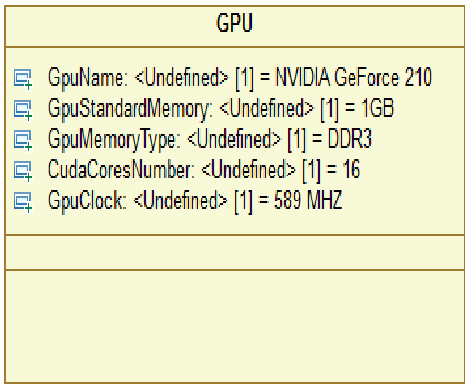

«component, hwComponent, hwProcessor»
Graphic Processing Unit
«component, hwComponent, hwComputingResource»
Streaming Multiprocessor
«component, hwCache, hwRAM»
L1 cache memory
«component, hwComponent, hwComputingResource»
Streaming Processor
«component, hwBus»
Bus
«component, hwDevice»
CUDA Core
«component, hwMemory»
Register Memory
«component, hwComputing...
Bus
«component, hwComponent, hwComputingResource»
Streaming Processor

Fig 5. Component diagram of GPU architecture

The properties of GPU compoenent are illustrated by a GPU class where all the GPU details are presented.


GPU

GpuName: <Undefined> [1] = NVIDIA GeForce 210
GpuStandardMemory: <Undefined> [1] = 1GB
GpuMemoryType: <Undefined> [1] = DDR3
CudaCoresNumber: <Undefined> [1] = 16
GpuClock: <Undefined> [1] = 589 MHZ

Fig 6. GPU class

5.1.2 Link between hardware architecture and software architecture

In order to treat the parallel execution of applications on GPUs we have chosen a simple algorithm of Arrays addition.

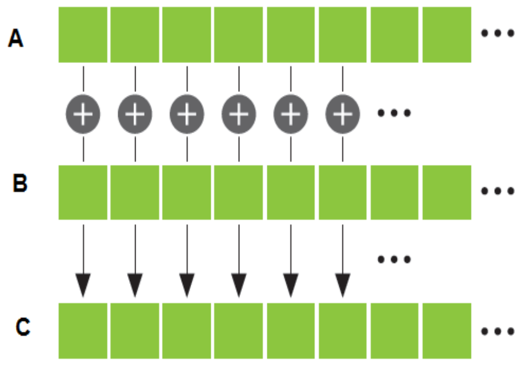


Fig 7. Vector Addition

The sequential algorithm of ***Vector addition*** of two vectors A, B (of N dimension) resulting a vector C is illustrated thereafter. It is necessary to run through a loop to execute the addition operation. So the time spent in executing vector addition is doubled.

```
Algorithm Vector Addition
Input:
A,B: array [1..N] d'entier
Output:C
Begin
  i: entier
  for (i=0 à N ; i++)
          C[i]= A[i]*B[i] ;
   Endfor
End
```

To optimize the runtime the vector addition algorithm can be executed in a parallel way on GPU architecture thanks to its multi-cores. Each addition operation of an element C[i] is calculated in a CUDA core basing the element A[i] and the element B[i]. [19] In this case the execution of vector addition follows these steps:

- Vector A loading on the CPU;
- Vector B loading on the CPU;
- Data (vector A, vector B) transfers from CPU into GPU;
- Calculating C: kernel parallel execution on GPU which is illustrated in the next algorithm;
- Data transfers (vector C) from GPU into CPU;
- Vector C display.

```
Algorithm Kernel
Input: A,B
Output:C
Begin
  int nmbr_bloc,nmbr_thread,affect;
  affect=nbre_bloc/nbre_thread;
  if (affect >=0) then
    Execute (Vector-addition (C[1], C[2],..., C[N])) ;
  endif
End
Vector-Addition (C[i])
Begin
    C[i] = A[i]+B[i] ;
End
```

The elemts C[i] are considered as blocs and executed in a parallel way using the A[i] and B[i] elements.

We specified the vector addition using MARTE profile then we allocated the application on the hardware architecture (CPU/GPU).

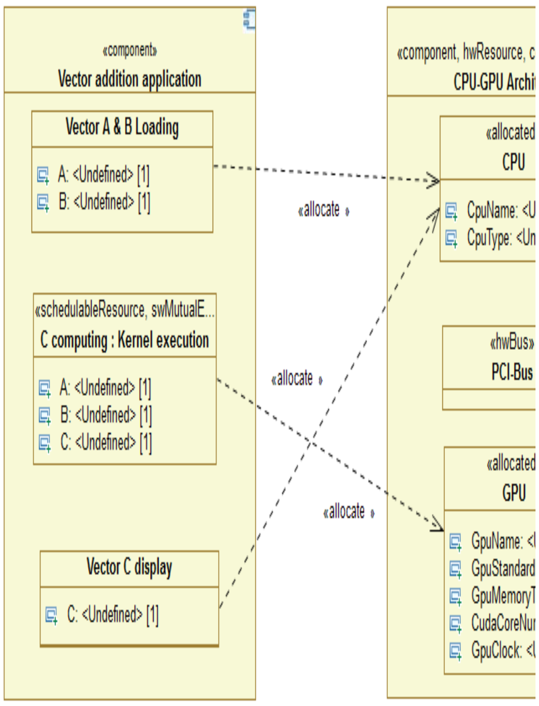


Fig 8. Allocation of vector addition application on hardware architecture

When a kernel is launched on GPU architecture only this kernel is executed, the other kernels will wait until it finishes to be executed. This notion is represented using MARTE stereotypes (swSchedulable, swMutualExclusion)

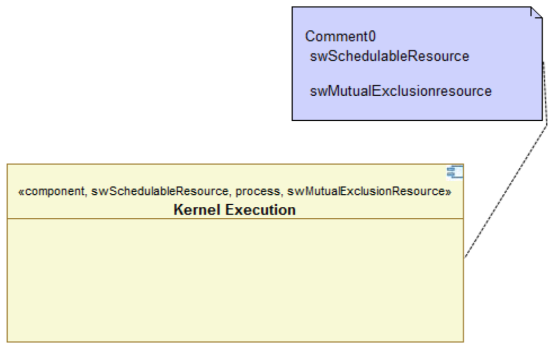

Comment0
swSchedulableResource
swMutualExclusionresource
«component, swSchedulableResource, process, swMutualExclusionResource»
Kernel Execution

Fig 9. Kernel execution stereotypes

5.1.3 State-transition diagram

When a task is launched a preliminary test is executed on the CPU to affect the task to the right processor. If it is a repetitive one it will be considered as a kernel which is going to be executed many times on the GPU-(SIMD) Single Instruction Multiple Data. If it is a sequential task, it will be run once on the CPU. The state transition events are **TimedEvent** and the state-trasition of task execution is stereotyped by **Timedprocessing** stereotype.

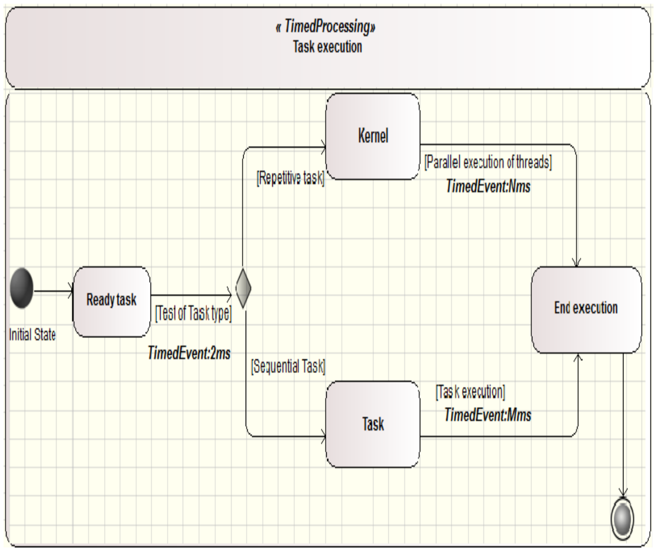


Fig 10. State-transition diagram of task execution

5.1.4 Timing diagram

When the Vector addition is launched the parameters (Vector A and Vector B) will be transferred into the GPU and the function of vector addition will be a kernel. Then on the GPU the kernel will be divided into blocs that will be executed in a semi-parallel way. Each bloc executes one operation of addition, then it is affected to the result vector C.

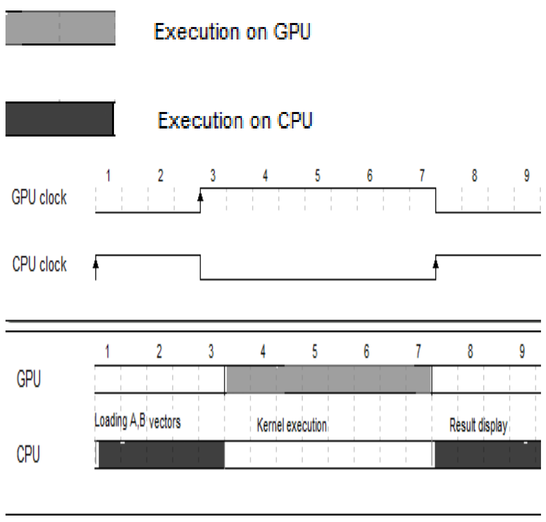


Fig 11. Timing diagram of vector addition on GPU

5.1.5 Sequence diagram

The process of vector additionis represented by a sequence UML diagram.

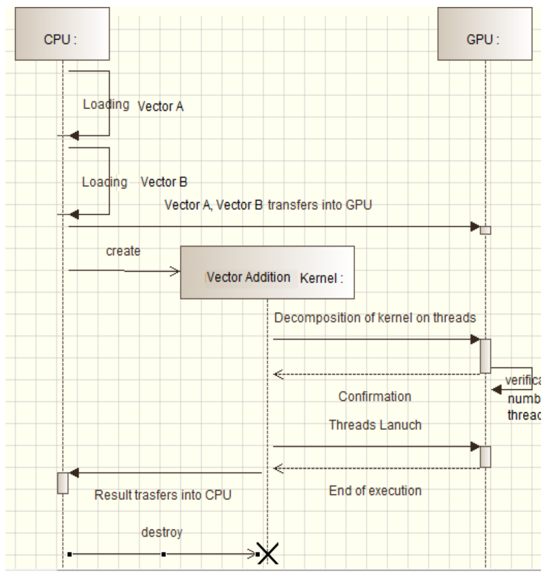


Fig 12. Sequence diagram of vector addition

5.2 Coupling MARTE with Event B

The approach of UML/MARTE transformation into Event B consists of representing the aspects of an application by UML/MARTE diagrams then they must be transformed into Event B specification and proved by Rodin. This technique uses MARTE as a start point for modeling oriented object models then they are proved and validated by Event B tools. Event B gives a correct semantic of the Graphic UML/MARTE models.

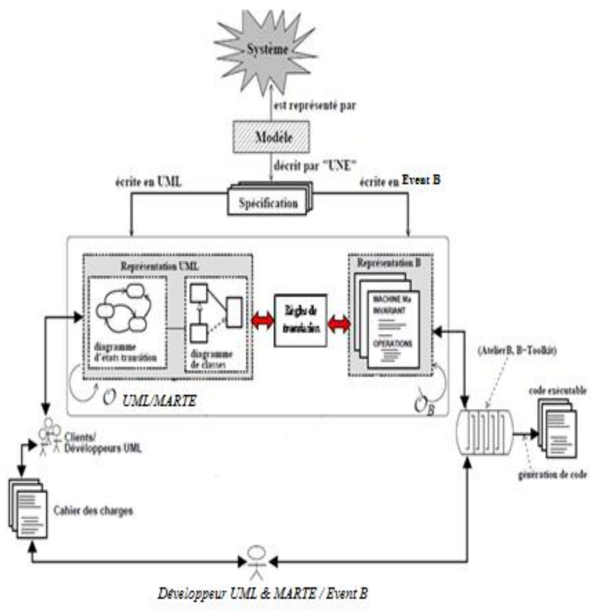
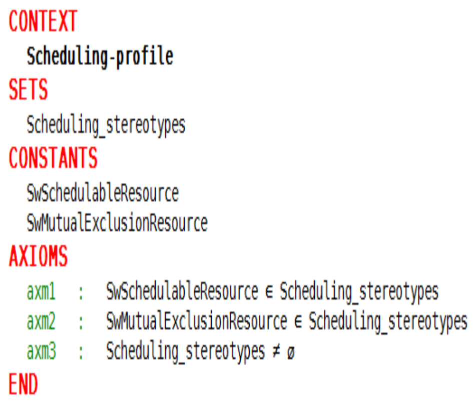


Fig 13. Process of MARTE trasformation into Event B

5.2.1 MARTE profiles' Instantiation

At the beginning we did a preliminary phase of MARTE's specification instantiation of MARTE Timing and scheduling profiles to Event B context from MARTE definition.

```
CONTEXT
   Timing-profile
SETS
   Timing_stereotypes
CONSTANTS
   TimedEvent
   TimedProcessing
   EventDuration
AXIOMS
   axm1   :   EventDuration ∈ N1
   axm2   :   TimedEvent ∈ Timing_stereotypes
   axm3   :   TimedProcessing ∈ Timing_stereotypes
   axm4   :   Timing_stereotypes≠ø
END
```



```
CONTEXT
  Scheduling-profile
SETS
  Scheduling_stereotypes
CONSTANTS
  SwSchedulableResource
  SwMutualExclusionResource
AXIOMS
  axm1  :  SwSchedulableResource ∈ Scheduling_stereotypes
  axm2  :  SwMutualExclusionResource ∈ Scheduling_stereotypes
  axm3  :  Scheduling_stereotypes ≠ ∅
END
```

5.2.2 Rules of MARTE specification transformation into Event B

In order to transform MARTE models into Event B specification we proposed a set of rules based on the state-transition, class diagram and Allocation diagram:

Rule 1: A class X is transformed into a Machine X.

Rule 2: The properties of class X are the variables of Machine X.

Rule 3: Each Machine X has a ContextX that defines its variables.

Rule 4: The states of state-transition diagram (of class X) are constants in the context ContextX.

Rule 5: The events of state-transition diagram are the events of Machine X.

Rule 6: The software Machines share the same context.

Rule 7: The operations of a class X are the events of the corresponding Machine X.

Rule 8: The hardware machine refines the software machines.

Applying these rules on the GPU MARTE specification we conclude the following Event B specification illustrated by a schema (cf.Fig).

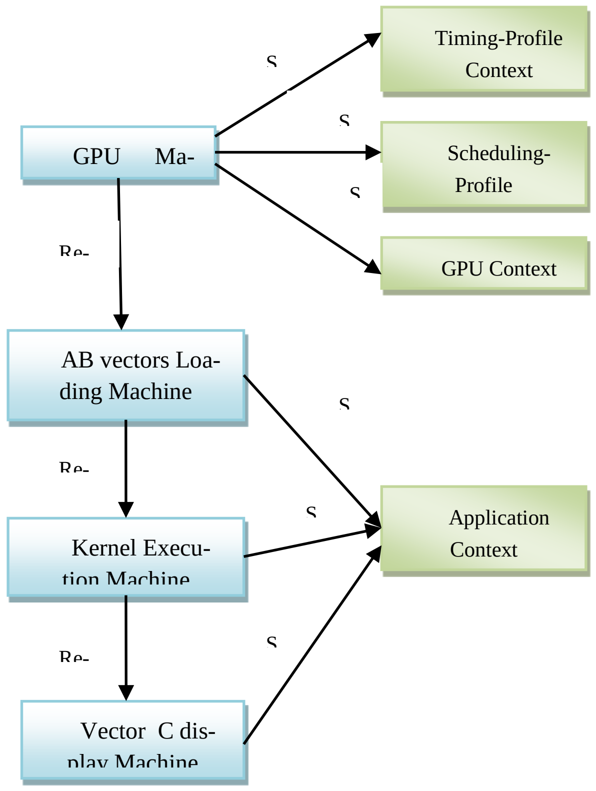


Fig 14. Resultant machines of transformation

The transformation of GPU class result a GPU machine using scheduling and timing profiles:

MACHINE

G PU

SEES

GPUContext

Timing-profile

Scheduling-profile

VARIABLES

Task-Type

Taskstate

Tstereo-type

Sstereo-type

Gpu-Name

GpuStandardMe-mory

GpuMemo-ryType

CudaCore-Number

```
    Gpu-
  Clock
    evdura-
  tion
INVARIANTS
    inv9      Task-
  :         Type ∈ TASKtype
    inv2      Taskstate ∈ St
  :         ate
    inv10      Tstereo-
  :          type ∈ Timing_stereotypes
    inv11      Sstereo-
  :          type ∈ Scheduling_stereotypes
    inv13      GpuStandardMemo-
  :          ry∈ℕ1
    inv14      CudaCoreNum-
  :          ber∈ℕ1
    inv15      Gpu-
  :          Clock∈ℕ1
    inv16      Gpu-
  :          Name∈Gpunamesetgpu
    inv17      GpuMemo-
  :          ryType∈MemoryType
    inv18      evdura-
  :          tion∈ℕ1
EVENTS
    INITIALISATION
  ≙
      ex-
    tended
        STAT
      US
      ordi-
    nary
BEGIN
      act11      Sstereo-
    :          type ≔ SwSchedulableResource
      act10      Tstereo-
    :          type ≔ TimedProcessing
      ac   3    Task-
       :       Type≔TR
      act4      Taskstate≔ReadyTa
    :          sk
      act12      Gpu-
    :          Name≔GeForce210
      act13      GpuMemo-
    :          ryType≔DDR3
      act14      CudaCoreNum-
    :          ber≔16
      act15      Gpu-
    :          Clock≔589
      act16      GpuStandardMemo-
    :          ry≔1
END

    TestE-
  vent  ≙
        STAT
      US
      or   in
```

```
      ary
ANY
        Tas
      kT
        Tst
      ate
WHERE
        grd1      TaskT=
      :          TR
        grd3      Tstate=ReadyTa
      :          sk
THEN
        act1      Taskstate ≔ Ker
      :          nel
        act2      Sstereo-
      :          type ≔ SwSchedulableResource
        act3      Tstereo-
      :          type ≔ TimedEvent
        act4      evdura-
      :          tion≔2
END

      Execu-
    tion   ≙
          STAT
        US
        ordi-
      nary
ANY
        Tst
      ate
WHERE
        grd1      Tstate = Ker
      :          nel
THEN
        act6      Taskstate≔ExecutionE
      :          nd
        act3      Sstereo-
      :          type ≔ SwMutualExclusionResource
        act4      Tstereo-
      :          type ≔ TimedEvent
        act7      evdura-
      :          tion≔30
END

END
```

For the application there are three machines refined from GPU machine which are presented in the next part:

```
MACHINE
    VectorABLoa-
  ding
REFINES
    G
  PU
SEES
    applicationCon-
  text
VARIABLES
    si
  ze



INVARIANTS
    inv1      size∈
  :          ℕ1
    inv7      k
  :          ∈ℕ
    inv12     A∈1‥size
  :          →ℕ
    inv13     ran(A)=ran(Arr
  :          ay)
    inv14     A∈ℕ
  :          ↔ℕ
    inv15     B∈1‥size
  :          →ℕ
    inv16     ran(B)=ran(Arr
  :          ay)
    inv17     B∈ℕ
  :          ↔ℕ
EVENTS
    INITIALISATION
  ≙
        STAT
      US
      ordi-
    nary
BEGIN
      act1      size:=10
    :          24
      act4      k
    :          :=1
END

    TableLoa-
  ding  ≙
        STAT
      US
      ordi-
    nary
WHEN
      grd1      k<size
    :          +1
THEN
      act3      A(k)
    :          :=k
      act4      B(k):=k
```

```
  :          ∗k
      act5      k:=k
  :          +1
END

END
```

```
MACHINE
    KernelExecu-
  tion
REFINES
    VectorABLoa-
  ding
VARIABLES
    si
  ze




    r
  est

    GpuCore-
  Number
    af-
  fec
    p
  os
INVARIANTS
    inv1      size∈
  :          ℕ1
    inv2      A∈1‥size
  :          →ℕ
    inv3      ran(A)=ran(Arr
  :          ay)
    inv4      A∈ℕ
  :          ↔ℕ
    inv5      B∈1‥size
  :          →ℕ
    inv6      ran(B)=ran(Arr
  :          ay)
    inv7      B∈ℕ
  :          ↔ℕ
    inv8      C∈1‥size
  :          →ℕ
    inv9      ran(C)=ran(Arr
  :          ay)
    inv  0     C∈ℕ
    :        ↔ℕ
     inv11    k
    :        ∈ℕ
     inv12    GpuCoreNum-
    :        ber∈ℕ1
     inv14    rest
    :        ∈ℕ
     inv15    i
    :        ∈ℕ
     inv16    af-
    :        fect∈ℕ
     inv17    pos
    :        ∈ℕ
EVENTS
     INITIALISATION
    ≙
          STAT
       US
        ordi-
      nary
REFINES
        INITIALISATI
      ON
BEGIN
        act4     size:=10
      :         24
        act1     k
                :=1
        act2     GpuCoreNum-
      :         ber:=16
        act3     rest:=10
      :         24
        act5     i
      :         :=1
        act6     pos
      :         :=1
END

     ThreadDevi-
    sion  ≙
          STAT
       US
        ordi-
      nary
BEGIN
        act1     af-
      :         fect:=rest÷GpuCoreNumber
        act2     pos
      :         :=k
END
     Compu-
    ting  ≙
          STAT
       US
        ordi-
      nary
WHEN
        grd1     af-
      :         fect>0
        grd2     rest
      :         >0
        grd3     i<pos+
      :         17
THEN
        act1     C(i):=A(i)+B
      :         (i)
        act3     k:=k
      :         +1
        act4     rest:=size
      :         −1
        act5     i:=i
      :         +1
END
END
```

5.3 CUDA Code generation

To exploit the Event B specification we propose a refinement approach to pass from Event B specification into a pre-code CUDA. The CUDA machine will be treated in another research work to generate an executable CUDA code which will be executed on the GPU architecture.

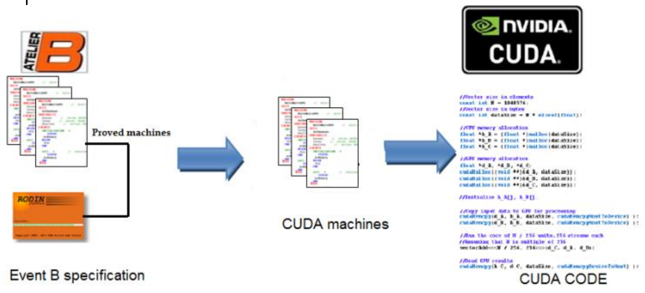


Fig 12.Trasformation process of Event B specification into Cuda code

This approach could be useful to people who can't implement parallel programs because it is difficult and it requires an experience in the field of programming. CUDA guaranties a parallel implementation of programs with specialized tools. In the case of vector addition algorithm the Event b specification will turn into the following Cuda Machine.

```
MACHINE
      CUDAMACHI
   NE
REFINES
      KernelExecu-
   tion
SEES
      CUDAcon-
   text
      applicationCon-
   text
```

```
VARIABLES
    devi-
  ceA
    de-
  viceB
    devi-
  ceC



    si
  ze
    Transfer-
  Mem
    CudaMal-
  loc
    af-
  fect
    GpuCore-
  Number
    r
  est


    p
  os
    Cudafr
  ee
INVARIANTS
    inv1      devi-
  :         ceA∈1··size→ℕ
    inv2      de-
  :         viceB∈1··size→ℕ
    inv3      devi-
  :         ceC∈1··size→ℕ
    inv8      Transfer-
  :         Mem ∈ TRANSFERDATATYPE
    inv10     size∈
  :          ℕ1
    inv11     ran(deviceA)=ran(Arr
  :          ay)
    inv12     ran(deviceB)=ran(Arr
  :          ay)
    inv13     ran(deviceC)=ran(Arr
  :          ay)
    inv14     devi-
  :          ceA∈ℕ↔ℕ
    inv15     de-
  :          viceB∈ℕ↔ℕ
    inv16     devi-
  :          ceC∈ℕ↔ℕ
    inv18     i
  :          ∈ℕ
    inv19     k
  :          ∈ℕ
    inv20     rest
  :          ∈ℕ
    inv21     GpuCoreNum-
  :          ber∈ℕ
    inv22     pos
```

```
  :          ∈ℕ
    inv24     Cudafree∈CudaFr
  :          ee
    inv25     CudaMal-
  :          loc∈Malloc
EVENTS
    INITIALISATION
  ≙
        STAT
      US
      ordi-
    nary
REFINES
      INITIALISATI
    ON
BEGIN
      act3      size:=10
    :         24
      act4      i
    :         :=1
      act5      k
    :         :=1
      act6      rest:=10
    :         24
      act7      GpuCoreNum-
    :         ber:=16
      act8      pos
    :         :=1
      act9      CudaMal-
    :         loc:=nonalloc
END

    CudaAlloca-
  tion  ≙
        STAT
      US
      ordi-
    nary
BEGIN
      act1      CudaMal-
    :         loc:=malloc
      act2      Cudafree:=nonFr
    :         ee
END

    CudaCPUtoGPUtras-
  ferts  ≙
        STAT
      US
      ordi-
    nary
WHEN
      grd1      CudaMal-
    :         loc=malloc
THEN
      act1      devi-
    :         ceA:=A
      act2      de-
    :         viceB:=B
    act3 :  TransferMem≔HostToDevice
END

  ThreadDivision ≙
    STATUS
    ordinary
BEGIN
    act1 :  affect≔rest÷GpuCoreNumber
    act2 :  pos ≔k
END

  CudaKernelLaunchADD ≙
    STATUS
    ordinary
WHEN
    grd1 :  affect>0
    grd2 :  rest >0
    grd3 :  i<pos+ 17
THEN
    act1 :  deviceC(i)≔deviceA(i)+deviceB(i)
    act2 :  k≔k +1
    act3 :  i≔i +1
    act4 :  rest≔size −1
END

  CudaGPUtoCPUtrasferts ≙
    STATUS
    ordinary
BEGIN
    act1 :  C≔deviceC
END

  CudaFree ≙
    STATUS
    ordinary
WHEN
    grd1 :  rest ≤0
THEN
    act1 :  Cudafree≔Free
END

END
```

After having refined our vector addition algorithm into a CUDA machine, our goal is to generate a valid CUDA code that guaranties the parallelism implementation on GPU architecture.

6. CONCLUSION

The paper suggests new approaches of specification and implementation of GPU SOC basing on MARTE models. The first approach consists on validating the proposed MARTE specification with the formal tool Event B. The second approach proposes to refines the event B specification to have a pre-Code CUDA.

The proposed approaches need to be improved by new rules. As a perspective we want to implement our approaches with automatic generation tools to apply our proposed rules of MARTE transformation into Event B specification and to validate task scheduling on the GPU architecture with formal tools such as Event B. Another perspective is to generate an executable code from the refined pre-code machine CUDA.

7. REFERENCES

[1] R. Laleau and A. Mammar, "An Overview of a Method and its support Tool for Generating B Specifications from UML NotationsC, In The 15st IEEE Int. Conf. on Automated Software Engineering, Grenoble (France), September 11-15, 2000.

[2] H. Ledang and J. Souquières, "Formalizing UML Behavioral Diagrams with B", In the Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics, Tampa Bay, Florida (USA), October 15, 2001.

[3]H. Ledang, J. Souquières & S. Charles, "ArgoUML+B : un outil de transformation systématique de spécifications UML en B", LORIA - Université Nancy 2, 2003.

[4] Antonio Wendell de O. Rodrigues & all. "A Modeling Approach based on UML/MARTE for GPU Architecture ", In the The Computing Research Repository (CoRR), May 2011.

[5] Antonio Wendell de O. Rodrigues, Frédéric Guyomarc'h and Jean-Luc Dekeyser, "Programming Massively Parallel Architectures using MARTE: a Case Study", In the Second Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011), 2011.

[6] Madeleine Faugère, Thimothée Bourbeau, Robert De Simone and Sébastien Gérard, "MARTE: Also an UML Profile for Modeling AADL Applications", In the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), 2007.

[7] Naoufel MACHTA M. Taha BENNANI Samir BEN AHMED, “MODELISATION ORIENTEE ASPECTS DES SYSTEMES TEMPS REEL”, In the 8th International Conference of Modelisation and Simulation (MOSIM 10), Hammamet (Tunisie), 2010.

[8] C. Andre, F. Mallet, M-A. Peraldi-Frati, “A multiform time approach to real-time system modeling Application to an automotive system ”, Int. Symp. on Industrial Embedded Systems, Lisoba, Portugal, pp 234-241, Jul 2007.

[9] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Philippe Soulard, Jean-Philippe Diguet, “A co-design approach for embedded system modeling and code generation with UML and MARTE”, Int Conf Design, Automation, and Test in Europe - DATE , Nice (France), pp. 226-231, 2009.

[10] Julio L. Medina and Alejandro Pérez Ruiz, “High level modeling for Real-time applications with UML & MARTE”, the 25th Euromicro Conference on Real-Time Systems (ECRTS’13), Paris, France, pp 13-16, July 2013.

[11] J.R Abrial, “The B-book: Assigning programs to meanings”, 1996.

[12] C. Métayer, J.-R. Abrial, L. Voisin. “Event-B Language”, May 2005.

[13] Yamine AIT-Ait-Ameur & all. “Vérification et validation formelles de systèmes interactifs fondées sur la preuve : application aux systèmes multi-modaux,” IN Journal d’Interaction Personne-Système, Vol. 1, No. 1, Art. 3, Septembre 2010.

[14] Joey Coleman, Cliff Jones, Ian Oliver, Alexander Romanovsky, and Elena Troubitsyna, “RODIN (Rigorous Open Development Environment for Complex Systems)”.

[15] Sylvain Collange, Yoginder S. Dandass, Marc Daumas, and David Defour, “Using Graphics Processors for Parallelizing Hash-Based Data Carving**”,** In *HICCS*, Hawaii International Conference on System Sciences, pp 1-10, 2009.

[16] Peter N. Glaskowsky. “NVIDIA’s Fermi: The First Complete GPU Computing Architecture”, 2009: http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA'sFermi-he_First_Complete_GPU_Architecture.pdf

[17] Sylvain Collange, Marc Daumas, David Defour & Régis Olivés, “Fonctions élémentaires sur GPU exploitant la localité de valeurs”, In SYMPosium en Architectures nouvelles de machines, Fribourg : Switzerland, pp 1-11, 2008.

[18] Reiji Suda and Da Qi Ren, “Accurate Measurements and Precise Modeling of Power Dissipation of CUDA Kernels toward Power Optimized High Performance CPU-GPU Computing”, In International Conference on Parallel and Distributed Computing, Applications and Technologies, pp 432-438, Higashi Hiroshima, Japan 2009.

[19] Jason Sanders, Edward Kandrot. “ CUDA par l'exemple ”, Pearson Education France, 2011.