Validating Timing And Scheduling Mar-te’s Profils Using Event B: Case Study Of A Gpu Architecture

Imane ZOUANE, Mostefa BELARBI, Chouarfia Abdellah
LIM Research Laboratory, University of Ibn khaledoun 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, Re-finements, 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 UML10/ 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 Le-dang [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 compartment diagrams to B specification. In addition, to complete his works, Ledang [3] has created a tool named Ar-goUML+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

---

10UML: Unified Modeling Language
Wendell [4][5] to generate OpenCL 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 Gradient 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 Gaspar2. Then an OpenCL valid code has been generated from Gaspar2 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 called 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 concretizes models and contexts as it is shown in the figure 2.

The Event B method is efficient because it uses tools like Atelier B\textsuperscript{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 and capacity of computing. [16]

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

---

\textsuperscript{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.

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,…).
Fig 5. Component diagram of GPU architecture

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

<table>
<thead>
<tr>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GpuName: &lt;Undefined&gt;</td>
</tr>
<tr>
<td>GpuStandardMemory: &lt;Undefined&gt;</td>
</tr>
<tr>
<td>GpuMemoryType: &lt;Undefined&gt;</td>
</tr>
<tr>
<td>CudaCoreNumber: &lt;Undefined&gt;</td>
</tr>
<tr>
<td>GpuClock: &lt;Undefined&gt;</td>
</tr>
</tbody>
</table>

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.

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.

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.
The elements C[i] are considered as blocks 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 (swSchedulableResource, swMutualExclusion).

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-transition 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
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 addition is 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 transformation 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.
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:

<table>
<thead>
<tr>
<th>MACHINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEES</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPUContext</td>
</tr>
<tr>
<td>Timing-profile</td>
</tr>
<tr>
<td>Scheduling-profile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task-Type</td>
</tr>
<tr>
<td>Taskstate</td>
</tr>
<tr>
<td>Stereotype</td>
</tr>
<tr>
<td>Stereotype</td>
</tr>
<tr>
<td>GpuName</td>
</tr>
<tr>
<td>GpuStandardMemory</td>
</tr>
<tr>
<td>GpuMemoryType</td>
</tr>
<tr>
<td>CudaCoreNumber</td>
</tr>
</tbody>
</table>
For the application there are three machines refined from GPU machine which are presented in the next part:
MACHINE VectorABLoading
REFINES G PU
SEES applicationContext
VARIABLES size

INVARIANTS
inv1 size∈\mathbb{N} : \text{size} \geq 1
inv7 k : k \in \mathbb{N}
inv12 A \in \mathbb{N} \cdot \text{size} : A \rightarrow \mathbb{N}
inv13 \text{ran}(A) = \text{ran}(\text{Arr}) : \text{ay}
inv14 A \in \mathbb{N} : A \rightarrow \mathbb{N}
inv15 B \in \mathbb{N} \cdot \text{size} : B \rightarrow \mathbb{N}
inv16 \text{ran}(B) = \text{ran}(\text{Arr}) : \text{ay}
inv17 B \in \mathbb{N} : B \rightarrow \mathbb{N}

EVENTS

INITIALISATION
\Delta

STAT
US
ordinary

BEGIN
act1 \text{size}:=10 : 24
act4 k : k:=1

END

TableLoading \Delta

STAT
US
ordinary

WHEN
\text{grd1} \ k<\text{size} : +1

THEN
act3 A(k) : i=k
act4 B(k):=k

END

MACHINE KernelExecution
REFINES VectorABLoading
VARIABLES size

INVARIANTS
inv1 size∈\mathbb{N} : \text{size} \geq 1
inv2 A \in \mathbb{N} \cdot \text{size} : A \rightarrow \mathbb{N}
inv3 \text{ran}(A) = \text{ran}(\text{Arr}) : \text{ay}
inv4 A \in \mathbb{N} : A \rightarrow \mathbb{N}
inv5 B \in \mathbb{N} \cdot \text{size} : B \rightarrow \mathbb{N}
inv6 \text{ran}(B) = \text{ran}(\text{Arr}) : \text{ay}
inv7 B \in \mathbb{N} : B \rightarrow \mathbb{N}
inv8 C \in \mathbb{N} \cdot \text{size} : C \rightarrow \mathbb{N}
inv9 \text{ran}(C) = \text{ran}(\text{Arr}) : \text{ay}
inv0 C \in \mathbb{N}
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.

![Transformation process of Event B specification into CUDA code](image_url)

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 CUDAMACHINE
  REFINES KernelExecution
  SEES CUDAcontext
```

```
END
END
END
```

```
EVENTS
  INITIALISATION
    STAT
      US
      ordinary
    REFINES INITIALISATION
    BEGIN
      act4: size:=10
        24
      act1: k
        := 1
      act2: GpuCoreNumber
        := 16
      act3: rest:=10
        24
      act5: i
        := 1
      act6: pos
        := 1
    END

  ThreadDeviation
    STAT
      US
      ordinary
    BEGIN
      act1: afeff:=rest=GpuCoreNumber
      act2: pos
        := k
    END

  Computing
    STAT
      US
      ordinary
    WHEN
      grd1: afeff
        := fact>0
```
### VARIABLES

- $\text{deviceA}$
- $\text{deviceB}$
- $\text{deviceC}$
- $\text{size}$
- $\text{Transfer}$
- $\text{Mem}$
- $\text{CudaMalloc}$
- $\text{alloc}$
- $\text{GpuCoreNumber}$
- $\text{rest}$
- $p$
- $\text{CudaFree}$
- $\text{ee}$

### INVARIANTS

- $\text{inv1: deviceA \in \mathbb{N} \land deviceB \in \mathbb{N} \land deviceC \in \mathbb{N}}$
- $\text{inv2: deviceB \in \mathbb{N} \land deviceC \in \mathbb{N}}$
- $\text{inv3: deviceC \in \mathbb{N} \land deviceA \in \mathbb{N}}$
- $\text{inv8: Transfer \land Mem \in \text{TRANSFERDATATYPE}}$
- $\text{inv10: size \in \mathbb{N}}$
- $\text{inv11: ran(deviceA) = ran(Arr)}$
- $\text{inv12: ran(deviceB) = ran(Arr)}$
- $\text{inv13: ran(deviceC) = ran(Arr)}$
- $\text{inv14: deviceA \in \mathbb{N} \land deviceB \in \mathbb{N} \land deviceC \in \mathbb{N}}$
- $\text{inv15: deviceB \in \mathbb{N} \land deviceC \in \mathbb{N}}$
- $\text{inv16: deviceC \in \mathbb{N} \land deviceA \in \mathbb{N}}$
- $\text{inv18: i \in \mathbb{N}}$
- $\text{inv19: k \in \mathbb{N}}$
- $\text{inv20: rest \in \mathbb{N}}$
- $\text{inv21: GpuCoreNumber \in \mathbb{N}}$
- $\text{inv22: pos}$

### EVENTS

#### INITIALISATION

- $\text{STATUSordinary}$

#### REFINES

- $\text{INITIALISATION}$

### BEGIN

- $\text{act3: size := 10}$
- $\text{act4: i := 1}$
- $\text{act5: k := 1}$
- $\text{act6: rest := 10}$
- $\text{act7: GpuCoreNumber := 16}$
- $\text{act8: pos := 1}$
- $\text{act9: CudaMalloc := malloc}$

### END

- $\text{CudaAllocation}$

#### STATUSordinary

### BEGIN

- $\text{act1: CudaMalloc := malloc}$
- $\text{act2: CudaFree := nonFree}$

### END

- $\text{CudaCPUtoGPUtransfers}$

#### STATUSordinary

### WHEN

- $\text{grd1: CudaMalloc := malloc}$

### THEN

- $\text{act1: deviceA := deviceB := deviceC}$
- $\text{act2: deviceB := deviceC}$
After having refined our vector addition algorithm into a CUDA machine, our goal is to generate a valid CUDA code that guarantees 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 refine 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


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


