SD2 ParaDIME Stack Initial Requirements Document

Initial Requirements for the ParaDIME Stack including: Devices (D2.1), Hardware Architecture (D3.1) and Runtime (D4.1).

For the success criteria and theoretical limits for the ParaDIME Stack including: Devices (D2.1), Hardware Architecture (D3.1), Runtime (D4.1) and Applications (D5.1), see SD1 Target Success Criteria.

For information regarding Applications Selection (D5.1), see SD3 ParaDIME Application Selection Document.

Document Information

Contract Number	318693
Project Website	www.paradime-project.eu
Contractual Deadline	Month 6 (30 March 2013)
Dissemination Level	Public
Nature	Report
Contributors	Anita Sobe (UNINE), Thomas Knauth (TUD), Oscar Palomar (BSC), Arindam Mallik (IMEC), Osman Unsal (BSC), Pascal Felber (UNINE), Wojciech Barczynski (AoTerra), Gina Alioto (BSC)
Reviewer	Thomas Knauth (TUD), Arindam Mallik (IMEC)
Keywords	Requirements, Metadata, Features

Notices:
The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the ParaDIME Project (www.paradime-project.eu), grant agreement n° 318693.

Change Log

Version	Description of Change
v1.0	Initial Draft released to the European Commission (based on internal v10).

Table of Contents

1 Introduction and executive summary

The SD2 ParaDIME Stack Initial Requirements Document includes a discussion of the initial requirements for each layer of the stack as originally proposed in the following documents:

- D2.1 Theoretical limits and target energy savings metrics success criteria document (Device)
- D3.1 Theoretical limits and target energy savings metrics success criteria and initial hardware architecture requirements document
- D4.1 Theoretical limits and target energy savings metrics success criteria definition and initial energy-efficient runtime requirements document
- D5.1 Theoretical limits and target energy savings metrics success criteria and applications selection document

The contents of these deliverables have been consolidated in order to present the requirements for the entire stack and thus, the entire project, in a more coherent and concise manner.

However, it should be noted that the document does not include the Target Success Critera for each of the stack layers, nor does it include information on applications selection. Instead, these criteria have been consolidated in separate deliverables, SD1 ParaDIME Stack Target Success Criteria Document and SD3 ParaDIME Application Selection Document.

2 The ParaDIME Stack at a Glance

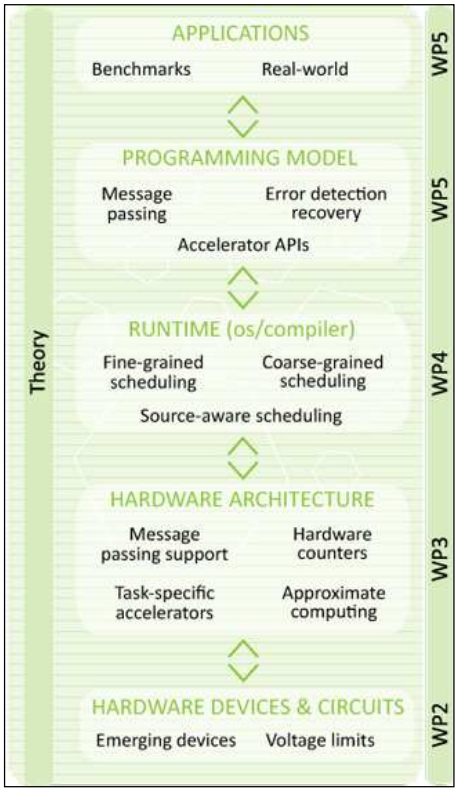

Figure 1: The ParaDIME Stack

The final outcome of the ParaDIME Project will be a full system stack that includes novel architecture components based on "beyond the state-of-the-art" technology, a programming environment and an execution framework for energy-efficient execution of concurrent applications. At the outset of the project, we define energy-efficient execution as a set of target energy savings metrics by which the success of the project will be measured during the evaluation phase.

Throughout the project, we will employ a series of methodologies as a way of attaining these target metrics. This document defines these methodologies as a series of ***functional requirements*** or ***feature*** in addition to ***metadata requirements*** to be implemented at each of the respective levels of the computing stack, device and circuit (WP2), architecture (WP3), runtime (WP4) and programming model levels (WP5).

3 Overview of methodologies and requirements for the entire computing stack

In the initial stages, ParaDIME will investigate requirements at the device and circuit (WP2), architecture (WP3), runtime (WP4) and programming model levels (WP5) of the computing stack that support the investigation of theoretical limits for energy efficiency.

We structure the document such that each layer of the ParaDIME stack corresponds to one section and relates to the work packages defined in the Description of Work. This structure allows us to differentiate the responsibilities for each part of the project. Furthermore, we differentiate between functional dependencies and metadata dependencies.

3.1 Functional Dependencies

We use this document to specify the functional dependencies between the layers. This is done by specifying the goals of each feature. Based on the goals we discuss first what this feature *provides* to other layers of the ParaDIME stack. Next, we list what each feature *requires* from other layers.

3.2 Metadata Dependencies

There are not only functional dependencies between the layers. To improve the features a layer provides, metadata from other layers are necessary. For example, the energy efficiency is highly dependent on the current application. If metadata about the characteristics of the application can be communicated between the layers, we claim that measures for energy efficiency can be applied more effectively.

We plan to forward metadata layer by layer as shown by the arrows in Figure 1. This avoids the need of interfaces between all layers and allows for metadata aggregation which lowers the communication efforts. The specific interfaces have to be well defined.

4 Programming model and API requirements (D5.1, precursor to Initial API due at month12)

The main idea is to create a software library/API that can handle and exploit different levels of concurrency on different levels of resources. The resulting software cooperates with the runtime deployment environment (see WP4). The API should be smart and cost-aware, i.e., considers communication costs (WANs at data center, fast network on node, shared memory on cores), energy costs, processing costs, performance (see guidelines by Intel [1]). The goal is to automatically deploy the software by considering the placement of data and the corresponding required transfer. Nonetheless, the programmer should be able to give hints for helping the API to decide on which actions should be performed. Thus, the API will provide different profiles based on hints from the programmer. Subsequently, monitoring results from the system, the programmer can decide which profiles should be applied. One of the main tasks of the API is to act as an interface between the programmer and the ParaDIME stack, i.e., forward and process application-specific data.

4.1 Annotations and Metadata

The actions performed by the API/programming model should be transparent to the programmer, but the developer should be supported in expressing *requirements* and *hints, e.g.*, complexity, parallelization possibilities and deadlines of the given program, to help the system to optimize towards the goals of the developer based on the given resources. Metadata reduces the runtime's complexity compared to a fully automated system. Also, allowing the programmer to incorporate domain-specific knowledge leads to better results for the application. The integration of the programmer will be realized by annotations and/or configuration files.

4.1.1 Provides

- *Annotations:* The API provides application related information to the other layers, which are provided by the programmer with the help of annotations and configuration files.
- *Metadata:* The programmer can specify deadlines, performance requirements, indicate critical sections and parts that can be easily parallelized. The goal is to provide a feedback loop between programmer and ParaDIME stack.

4.1.2 Requires

- *Statistics and state*: Information is required from the lower layers regarding available resources and current energy consumption to improve the energy efficiency on the API level,. The energy consumption should be aggregated high level data from the lower layers, and will be further used to predict possible measures and profiles, or to inform the user about current energy efficiency.

4.2 Efficient Message Passing

Applications based on the API will run transparently on distributed and heterogeneous resources. We target this goal by exploiting the natural principles of message passing. This allows for higher flexibility regarding types of resources without relying on shared memory.

4.2.1 Provides

- *Libraries, Frameworks:* The API will provide the necessary libraries and frameworks to support the programmer and the other ParaDIME layers to efficiently use the methodologies of message passing. Given the support from the hardware and runtime, the API enables programs to run more efficiently on distributed and non-distributed resources.

4.2.2 Requires

- *Hardware support:* To improve the efficiency of message passing, the API requires hardware support, e.g., by avoiding the network stack if running on local nodes.

4.3 Operation below safe V_{dd}

Since we assume voltage and frequency scaling at the CPU level, we assume a higher error probability that can even influence the application itself [2]. The API enables the programmer to give hints about compensation actions if errors occur or indicate how

many errors are acceptable before a compensation action becomes necessary. We plan to support different measures on three levels: error detection, error recovery and approximate computing.

4.3.1 Error detection

It is possible to employ many different error detection mechanisms, with different advantages and disadvantages regarding error detection capabilities and overheads. Besides the hardware and its own error detection mechanisms, error detection capabilities are highly application dependent.

Depending on what the goal of the application is, the programmer should have the possibility to select what error detection mechanism will be applied. To reach this goal profiles of error detection mechanisms are needed to provide predictions of the final outcome.

Provides

- *Error detection profiles annotations:* The API allows the programmer to indicate what error detection should be applied with the help of annotations.
- *Metadata:* The chosen error detection parameters and metadata will be provided to the other layers like what error detection mechanism is currently applied.
- *Statistics:* Furthermore, the current state of the error detection will be forwarded to the other layers.

Requires

- *Statistics and current state:* There should be a feedback loop between API, the runtime and hardware regarding the current state of the error detection on each of the layers. Failure statistics improve the quality and help to adapt the error detection mechanisms if needed.
- *Current voltage and frequency settings:* Alternatively, information on current voltage and frequency state can be used to adapt the error detection mechanism automatically.

4.3.2 Error recovery

As shown in [2] lightweight transactional memory (TM) implementations can be used to rollback after error detection. If we concentrate on error recovery instead of thread synchronization, the typically high overhead of TM can be avoided. Additionally, we target a TM with hardware support. Furthermore, the programmer should be able to indicate the areas that should be covered by the TM.

Provides

- *Error recovery profiles:* The API provides libraries, frameworks and needed annotations for TM support. Further it provides error recovery metadata to the other layers like size of the transaction, recovery statistics, and hints from the programmer regarding scope of the transaction and the error recovery profile chosen.
- *Metadata:* Any information on current error recovery mechanisms, annotations on critical and non-critical sections are forwarded via the runtime layer.

Requires

- *Hardware support:* We want to avoid any overhead introduced by TM, therefore we require hardware support for TM and the related statistics on the hardware side.

4.4 Approximate computing

To reduce the overhead of error detection and recovery, we envision the possibility of approximate data types [3]. Many applications do not rely on precise calculations, which can be exploited. By letting the programmer define approximate data types (which are supported by hardware) it is possible to reduce error detection overhead and improve energy efficiency.

4.4.1 Provides

- *Annotations:* The API provides necessary annotation to enable the indication of approximate data types.
- *Metadata:* Any metadata related to the annotations will be forwarded to the other layers.

4.4.2 Requires

- *Hardware support:* To enable approximate computing, the hardware has to support this functionality.

4.5 Adaptability

The actions within the API have to adapt to the system's state (i.e., distributed, parallel, single-core, failures, energy consumption, requirements, etc.), with or without informing the programmer and/or the application.

4.5.1 Provides

- *Statistics and state information*: The API will provide metadata to the other layers about changes in profiles regarding error detection, error recovery, message passing and transparency.
- *Energy Profiles*: Depending on the information provided by the programmer and the other ParaDIME layers, energy profiles will be defined, controlled and monitored. Automatic and semi-automatic decisions on the API level (e.g., regarding error detection/correction, message passing and approximate computing) will be executed.

4.5.2 Requires

- *Monitoring*: A mechanism is required to monitor the current system's state and check if the requirements set by the programmers are met. This is necessary to automatically (and semi-automatically) adapt the functionality of the programming model. The following parameters will be monitored:

Parameter	***Comments***
Base Energy Consumption	Derive from other metrics [4], and/or apply energy model, such as introduced in [5]

Performance	Execution time, Data Migration/communication costs
QoS metrics	Focus on dependability - replication, placement, etc.
Costs (money) for using the resources	
Failure statistics	Current failure probability
Currently available resources	Processing, memory

5 Software runtime requirements (D4.1)

While the programming model and API set the framework within which the program must be expressed, the runtime executes the program. The runtime encompasses all software components which run alongside the user application. We do not intend to provide a runtime in the sense of, for example, a Java virtual machine environment. Rather, the runtime is a collection of user space libraries, daemons, and operating system enhancements. Because ParaDIME's focus is on energy efficiency the runtime employs six mechanisms to reduce the application's overall energy consumption. We outline the requirements and features of each mechanism below.

5.1 Operation below safe V_{dd}

We want to increase the energy-efficiency at the CPU level by decreasing the CPU's supply voltage. Modern CPUs already incorporate energy-efficiency measures. The processor supports several *power states*. The ACPI standard defines exactly four different power states: C0 - C3. Besides power states, the processor may also support different *performance states*. The number of performance states differs between processors. They are numbered P0, P1, …, Pn. Each successively higher state reduces the processors performance, because the voltage and/or frequency are reduced. Usually the operating system exclusively controls power management features. However, by overriding the operating system it is possible to scale voltage and/or frequency even more aggressively, enabling even higher savings...

5.1.1 Provides

- *Voltage and frequency scaling:* The runtime provides automatic voltage and frequency scaling and allows for low and near threshold operation. It will provide the current state information to the application.

5.1.2 Requires

- *Information on critical sections:* The runtime requires the application code to be annotated and divided into critical and non-critical sections. During a critical section the supply voltage must not be decreased to guarantee fault free execution, e.g., inside the operating system's kernel.
- *Error detection and handling:* It requires that compensation mechanisms exist, either at the application or runtime layer, to cope with errors due to supply voltage changes. For the runtime to decide when to change the supply voltage it requires a model of the induced costs, i.e., error checking overhead, error frequency, and recovery costs.

- *Hardware support for error detection:* We expect that the error rate increases as the supply voltage decreases; hence error detection mechanisms are required.
- *Voltage setting support:* The hardware should support manual supply voltage setting. The application software must be amended with fault detection mechanisms.
- *Information on current power consumption:* The model also requires information from the hardware layer about the relationship between supply voltage and power consumption to decide when it is useful to reduce the supply voltage

5.2 Higher energy-efficiency at the data center level

To increase the energy efficiency at the data center our goal is to increase average utilization levels. By pushing utilization levels up, the comparatively high baseline power consumption is compensated for. We plan to combat the previously mentioned drawbacks of high utilization levels by executing a mix of compute tasks on each server. We distinguish between two types of tasks: *interactive* and *batch*. Interactive tasks have stringent performance requirements expressed as service level agreements (SLAs). They are latency-sensitive because they are end-user driven. A typical example is a web server. The end user requests a web page from the server which should answer as quickly as possible, e.g., within one second. Batch tasks, on the other hand, have turnaround times 2 to 3 magnitudes larger than interactive tasks, i.e., hours or days. This flexibility allows us to achieve utilization values of 90% and higher. Each server executes a mix of interactive and batch tasks. We have to ensure that interactive jobs never account for more than, say, 50% of the load. Additional capacity is consumed by batch tasks. Whenever there is a spike in interactive load, batch tasks will yield their resources to the interactive tasks. As soon as the surge in interactive load subsides, the batch tasks will continue executing, occupying all available spare resources.

5.2.1 Provides

- *Scheduling decisions:* The energy efficient runtime provides scheduling decisions based on the information given to it by the hardware and application. Based on the application mix the runtime assigns applications wrapped in a virtual machine container to physical machines (servers).
- *Migration:* The runtime also provides support to migrate applications between physical servers if it deems this beneficial with respect to energy efficiency.

5.2.2 Requires

- *Type of application:* At the data center level we require applications with complementary resource requirements. Each application must fall into one of two categories: batch or interactive. Each server will host a mix of batch and interactive applications to increase its overall utilization. Batch applications are less sensitive to temporary resource shortages and have flexible deadlines. It is required that the application type is available to the runtime, either via static annotations or dynamically through an API.
- *Application requirements:* Further, the application's deadline and budget (e.g., in Euro) are also required by the scheduling runtime.

- *Application Mobility:* Scheduling at the data center level requires that the schedulable entity is a virtual machine. Applications must be pre-packaged in the form of a virtual machine image which is instantiated by the runtime system.
- *Status Information:* To achieve energy efficiency at the data center level we require information from the runtime about the current resource utilization of, among others, the CPU, RAM, disk, and network. The current utilization is required for each application and server. The utilization bounds the achievable consolidation ratio.

5.3 Energy-proportional computing at the data center level

Energy-proportional computing is a concept where the power required by a computing system is directly proportional to the work done. A standard commodity server is typically **not** energy-proportional. Earlier, we introduced the example of a server which draws 120 W at 0% utilization. An energy-proportional server would draw 0 W at 0% utilization. The power drawn would increase linearly with utilization. Even though individual components, here servers, may not be energy proportional, it has been shown that energy-proportionality can be approached at the aggregate level. A server with close to 0% utilization can be switched off, while the work is taken over by the remaining servers.

Turning components off, however, is complicated by dependencies to the state stored on the server. Large amounts of data may be cached in memory or stored on disk. Turning the server off renders this state inaccessible. Any potential solution to turn off servers must work around the “state problem”.

5.3.1 Provides

- *Placement:* The runtime provides energy-proportional placement decisions of virtual machines.
- *Main unit control:* The runtime decides which servers to switch off and which workloads must be moved between servers. Migration decisions follow a cost/benefit analysis.

5.3.2 Requires

- *Ability to migrate*: In addition to the requirements outlined in Section 5.2 the runtime requires that it be possible to migrate the applications. Migration is necessary to consolidate workloads during periods of low utilization. Migration must only be supported within a data center.

5.4 Carbon-aware scheduling on the inter-data center level

Energy efficiency is less important if sufficient cheap and emission-free energy sources are available. Because energy is a growing cost factor for data center operators, reducing the overall consumption in turn reduces the overall operating expenditures. Coupled with penalties for carbon emissions the urge to cut energy consumption is even stronger. If, however, a cheap and “green” energy source is available, the overall consumption may suddenly be secondary. When data centers have access to alternative energy sources, say solar and coal, the

question of where to process a task is then also dependent on where energy is cheap, plentiful, and green.

5.4.1 Provides

- *Distributed Scheduling:* The runtime provides scheduling decisions across data centers in contrast to the tasks described in Section 5.2 and 5.3 which are concerned with scheduling within a data center. The placement decision is based on information about projected energy availability, cost, and heat demand.

5.4.2 Requires

- *Restartability:* If possible, tasks must be restartable, i.e., they can be stopped in one data center and started again in a different data center with only minimal service interruption. Restartability is facilitated by applications which have no/little state or where the state can be reconstructed (soft state).
- *Statistics:* The runtime requires information about the predicted availability of energy sources and the predicted price from the hardware.

5.5 Heterogeneous Computing

CPUs are general purpose microprocessors. But even they have a multitude of special purpose circuitry to help with, e.g., floating point operations and streaming data manipulation (SSE1/2/3/4). Besides the CPU, there are other components, which take over specialized tasks. The most prominent example is the graphics processing unit (GPU). The GPU is an *accelerator* for graphics processing. Rendering 3D scenes is a complex task which can, however, be sped up significantly with special-purpose hardware. The idea of accelerators is to implement certain functionality in hardware instead of executing it in software on a general purpose processing unit. By offloading tasks to the accelerator, the general purpose unit is free to do alternative work, or sleep if there is nothing else to do. The accelerator, because it is specialized, will perform the same task more efficiently.

5.5.1 Provides

- *Acceleration*: special purpose computations are performed on tailored hardware. Multiple accelerators may exist to speed up alternative parts of the computation.

5.5.2 Requires

- *Application kernels*: for acceleration to be beneficial the offloaded code fragments must be small and performed frequently.
- *Annotations*: the programmer must mark code fragments eligible for offloading. The annotation must also include the type of accelerator to use if more than one accelerator is available.

5.6 Energy-efficient Storage System

The energy-efficient storage system is an object store with a simple interface to get, put, update, and delete objects. Objects are binary data blobs as far as the storage system is concerned. Each object is replicated R times, where R is the replication factor. The replication factor is tunable. It allows different trade-offs for data

availability and storage overhead. Besides the minimum replication factor R, there exist additional copies of popular objects. These exist solely to cope with increased read requests. Whenever the aggregated client read throughput exceeds the available bandwidth of live replicas, additional copies are brought online. This ensures that the storage system only consumes energy in proportion to the client demands.

5.6.1 Provides

- *Storage API:* The energy-efficient storage system provides an interface to the application to persistently store data. The interface is linked to the application in the form of a library. The library provides the basic primitives to create, read, update, and delete data objects. To the library each object is an opaque binary string. The storage library provides an additional level of encapsulation: the details of accessing the storage system can be changed without re-writing the dependent applications.

5.6.2 Requires

- *Storage API interfaces:* The energy efficient storage system requires that applications are written against the API that the storage system provides.
- *Disk control:* At the hardware level, the storage system requires that individual disks can be powered off.
- *Storage statistics:* The storage system requires feedback from the runtime about the read and write throughput observed with the current workload.

6 Hardware architecture requirements (D3.1)

The main goal of this work package is to develop the hardware architecture layer of the ParaDIME stack. The architecture will target minimization of energy and will incorporate support for the ParaDIME methodologies.

6.1 General remarks

The following measures enable us to minimize energy consumption, but are not directly connected to any of the ParaDIME methodologies. The microarchitecture of the processor will not include aggressive techniques for speculation since they are known to be very power-hungry and not energy-efficient. Therefore, branch prediction, out-of-order scheduling and memory-aliasing prediction will be kept to a minimum or even not used at all.

6.2 Efficient Message Passing

At the hardware level, we will leverage the opportunities provided by using message passing to eliminate structures and overheads that are necessary for shared-memory systems. For example, the hardware can be greatly simplified if there is no need for cache coherency protocols.

On the top of that, the hardware will implement mechanisms to improve the energy efficiency of message passing. One goal is to reduce the overhead of sending messages, by providing ways to avoid the network stack. One possibility is to introduce new instructions to send data to a specific level of the cache hierarchy of the destination core. We will also provide fast context switch for small tasks that will run

on a co-processor. The data that the task will consume will be sent through a special register file which is shared between the main core and the co-processor.

Another goal is to reduce the amount of messages or data to send. For distributed data structures where gathering the required data can be very costly in terms of energy, we will provide a mechanism to send the task where the required data is. In order to support inter-core communication for cores in the same chip with shared memory, we will provide support to leverage the translation lookaside buffer (TLB), where TLB entries are modified instead of moving data and implement copy-on-write to the TLB. This will reduce data movement and improve performance.

6.2.1 Provides

- *Message passing hardware support:* Efficient message passing can be done if overheads can be avoided, i.e., we provide a mechanism to avoid the network stack
- *Task switch:* We provide a fast context switch for small tasks. Further improvements can be reached by a mechanism for "Task passing" that executes code where data is.
- *Task mobility:* We support the sending of complex data structures with a single request.

6.2.2 Requires

- *No shared data:* To support efficient task handling we require that the application runs without simultaneous sharing of data. This should be ensured by the programming model layer or by the runtime
- *Metadata:* In order to support fast task switching, we need information on the size of tasks. This can be done by annotations at the API layer or estimated by the runtime.

6.3 Operation below safe Vdd

Reducing the supply voltage of a circuit (Vdd) is a well-known technique to trade performance for power. However, its applicability is limited by safeguard bands to ensure correctness. Redundancy can be used to detect and correct errors but full replication incurs significant energy overhead.

We will provide error detection and correction mechanisms to ensure correct execution when V_{dd} is below the safe limits. One possible mechanism that we will investigate is hardware transactional memory, since its ability to keep a checkpointed state is very handy for error correction. Also, we will utilize replicated transactions for high error detection capability in order to reduce the V_{dd} to considerably lower level. Replication will be selective, only when it makes sense to lower the V_{dd}, thereby limiting the overhead of replication.

Another possible energy-efficient technique is supporting redundancy in memory structures to reduce the V_{dd} for caches. For example, we will design a data cache that is able to keep multiple copies of data but only when V_{dd} is below the safe limit.

6.3.1 Provides

- *Voltage control:* We provide the means to control and change the V_{dd}

- *Metadata:* We will provide information on the minimum V_{dd} at which the processor can operate ensuring correctness.
- *Error Detection:* To ensure correct execution we will provide mechanisms and support for detecting and correcting errors due to the lowering of V_{dd}.

6.3.2 Requires

- *Voltage control information:* Runtime is responsible for indicating when to lower the V_{dd} and by how much.
- *Application metadata:* Provides mechanisms to indicate when V_{dd} can be lowered. E.g., by indicating non-critical sections
- Error probability: The device layer provides information on probability of error at a given V_{dd}. This information will be used for error detection mechanisms and will be forwarded as high-level information to the upper layers.

6.4 Approximate computing

The hardware architecture will incorporate several techniques for approximate computing. It can implement smaller or faster ALUs for reduced floating point precision and known narrow values. It also can store fewer bits in the register file for these data, thereby reducing energy consumption. In a similar fashion, data caches and memory will be modified to exploit the smaller size of data, either by powering down some blocks or packing more data in the same capacity.

ECC is a very power-hungry mechanism that can be simplified or partially removed if results of a computation don't require the highest possible accuracy. This can be combined with a more aggressive lowering of the V_{dd}.

We will also investigate the use of neural network accelerators to approximate computation with high energy-efficiency.

6.4.1 Provides

- *Reduced Precision*: ALUs, register files that exploit "known" narrow values and reduced FP precision.
- *Reduced Data:* A mechanism to exploit reduced data width in data cache and memory.
- *Reduced Error Correction Coding:* Reduced/removed/simplified ECC.
- *Operation below safe V_{dd}*: Aggressive operation below safe V_{dd}, with error detection or correction.
- *Accelerator support:* Approximate computation based on neural network accelerators.

6.4.2 Requires

- *Annotations:* The API/runtime should provide mechanism to annotate when data is narrow/reduced FP precision.
- *Non-critical sections:* We do not only support the use of approximate data types, but also provide approximate functions. Therefore the API has to provide a mechanism to annotate when one function can tolerate errors and can be approximated.

- *Error Probability:* For error detection and monitoring of approximate functions and data types we require information about error probabilities at a given V_{dd}.

6.5 Heterogeneous computing

The use of specialized accelerators is one of the most promising ways to deal with the dark silicon problem. With specialized logic, a lot of overheads of general purpose processors are eliminated, with great energy savings and improvements in performance. We will focus on accelerators for message passing e.g. support to accelerate sending complex data patterns and error detection. We will also provide "generic" accelerators such as vector units, which can be used for any task that has DLP. For approximate computations, we will consider also the use of neural network-based accelerators.

6.5.1 Provides

- *Accelerators:* We provide specialized accelerators for message passing and error detection, vector accelerators and neural network-based accelerators.
- *Main unit/core control:* While using accelerators we provide mechanism to switch off the main unit for further energy savings.

6.5.2 Requires

- *Metadata:* To indicate whether a task can be sent to an accelerator the API should provide mechanisms to offload tasks to the accelerator, e.g., by annotations.
- *Accelerator control:* Based on the information from the application and its own metadata the runtime decides whether to use accelerators.
- *Unit/core control:* We require information whether the currently used units/cores should be switched off, which also affects accelerators. The runtime is responsible for providing this information.

7 Device requirements (D2.1)

The main objectives of this work package are to define, implement and evaluate a compact model and subsequently a device plug-in block for advanced technology devices across different logic architectures. The results obtained from this work-package will demonstrate the effect of technology scaling on energy efficiency. The plug-in block will allow a run-time architectural simulation of advanced devices.

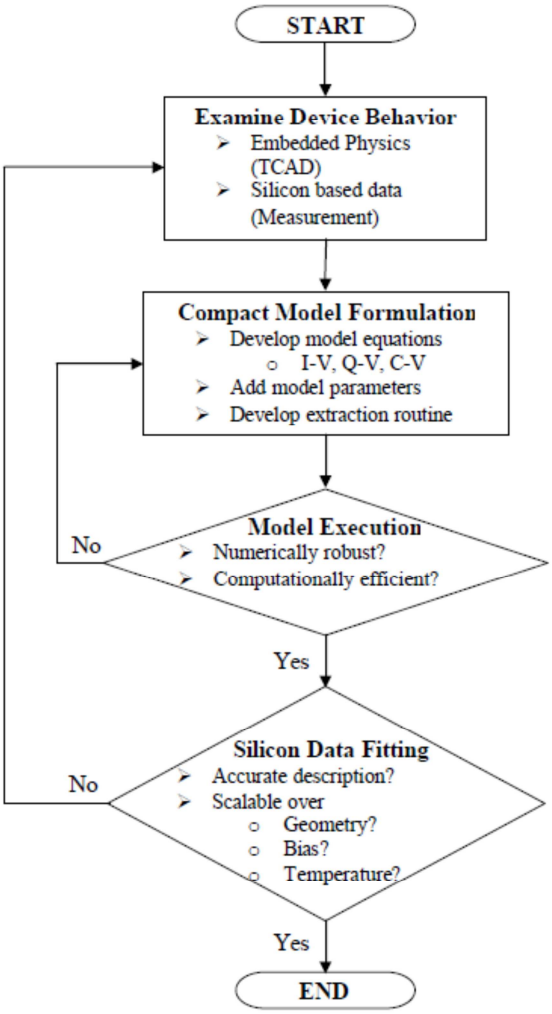

Figure 2: ***Methodology for compact model development***

7.1 Compact model

A compact model is a mathematical relation to depict the complex device physics in the transistor. An accurate model stemming from physics basis helps technologists and circuit designers to foresee benefits of technology scaling beyond the available silicon data. Figure 2 depicts the methodology for compact model development. We will focus on the following points.

- *Analysis of device behavior:* The compact model tries to mimic the actual physical behavior of the transistor as closely as possible. To enable that, accurate Technology Computer Aided Design (TCAD) simulations serve as a starting point. Additionally, measured data from state-of-the-art devices helps to examine the accuracy of the model as well as introducing new physical phenomenon.
- *Analytical formulation:* The primary objective of the compact model is to derive mathematical formulation to capture physics of the device from TCAD simulations. In a real device, quantities such as the doping profiles, junction depths, etc. are very complex. The effect of quantities such as these, physics and technology related model parameters are added to the compact model in order to precisely get an estimate of their effects on device behavior.

- *Robustness and flexibility of the model:* A compact model without enough flexibility and without the ease of parameter extraction is virtually useless for real world circuit design. Once the model equations are formulated, the next step is to examine the numerical robustness of the model. The model should not have any convergence issues and should be computationally efficient. To satisfy this constraint, physics based equations modified to accelerate the model computation.
- *Accuracy of the model:* The accuracy of the compact model is fittings its outcome against silicon data. Successful description of silicon data over different geometry, bias and temperature marks the completion of model development.

7.2 Methodology for device-simulator interface

The interface between compact model and architecture simulator implemented in WP2 should ensure that parameters for the technology devices can interact with the simulator in an efficient way. The task will involve a circuit-level analysis for these devices using the developed compact model. Based on the analysis, we will estimate the energy consumption at logic block level (ALU). Furthermore, we will explore the behavior of a circuit below the safe V_{dd} limits under this task. The objective of such exploration would be to understand the probability of operational correctness when we push the V_{dd} limits beyond the safe operating conditions, as defined by traditional circuit design principle.

7.2.1 Provides

- *Future Technologies:* Viable alternatives for devices in near and far-future technology nodes.
- *Metadata:* Device level characteristics for transistors at technology nodes in sub 14nm based on compact model.
- *Analysis:* Power-performance analysis of Characteristics of advanced technology devices at different supply voltages (V_{dd})
- *Error Probability:* Probability of an error for simple representative logic blocks (e.g. Ring-oscillators, ALU)
- *Interfaces:* Device-simulator plugin block in the form of a look up table (LUT) to transfer device level characteristics to the architectural simulator developed in WP3.

7.2.2 Requires

- *Information on supply voltages:* To give proper information on error probabilities, the range of supply voltages at which the system components would be run is required from the architectural simulator.
- *Simulation metadata:* The SPICE level netlist is needed to simulate a circuit block behavior using the advanced technology devices. As different circuit blocks have various resiliencies to errors, it is important to look at the effect of below safe V_{dd} operation for a few of them.
- *Test-bench:* A test-bench for the logic blocks is required, capturing different application level scenarios. This test bench will be used to evaluate the error probability of a circuit block.

8 Bibliography

[1] B. Steigerwald and A. Agrawal, "Developing Green Software," Intel Technical Report, 2011.

[2] C. Fetzer and P. Felber, "Transactional Memory for Dependable Embedded Systems," in *HotDep*, 2011.

[3] A. Sampson and e. al., "Enerj: Approximate data types for safe and general low-power computation," in *PLDI*, 2011.

[4] G. Bhavishva and e. al., "Portable, scalable, per-core power estimation for intelligent resource management," in *Green Computing Conference*, 2010.

[5] T. Rauber and G. Rünger, "Modeling the energy consumption for concurrent executions of parallel tasks," in *CNS*, 2011.