Operating Environment on-line Metrics for Application Architecture Matching

Milad El Khodary, Jean-Philippe Diguet, Guy Gogniat
LESTER, CNRS / UBS Research center. 56321 Lorient Cedex. France

Abstract

The need to execute highly demanding and real-time applications on ambient computing networks will be a big challenge in the future. In this paper we present an Operating Environment (OE) that allows application partitioning on such platforms. The OE decisions are made based on a trade negotiation protocol and energy efficiency and resource matching metrics. The metrics matches application and architecture using purely application and architectural defined characteristics. Application is modeled as hierarchical task graphs with Meta-data and architecture as a reconfigurable and multi-granular instruction set virtual processor. Metrics complexity is adequate for on-line decisions thanks to easy generalization through the application hierarchy.

Keywords: adaptive systems, ambient computing, metrics, operating environment

1 Introduction

This work is a part of the Æther[1] project which investigated auto-adaptation approach for future emerging electronic systems and applications. The project has identified the need to execute highly demanding and real-time applications on ambient computing networks. Projects like java-RT [1] has already emerged from strong today's need to execute hard or soft real-time applications on broadly available systems. Taking this a step further, future applications will be distributed on widely available ambient computing platforms as today's high intensive application benefits from grid systems. But adapting dynamically available computing resources to application objectives will be a great challenge in future emerging systems. In fact, application computing performance objectives could undergo huge variations during the life cycle of the application and the amount of available computing resources in certain urban area is dynamically variable.

Today's OS will be extended as Operating Environment (OE) to fit to the context of pervasive computing and will guarantee an acceptable efficiency for the whole platform. This could be possible through the characterization of: application objectives and quality of services; architecture computing performance and then matching them at runtime. The specification of applications and architectures is done at design time while matching and auto-adaptation is done by the OE at runtime. Section 2 presents the Æther auto-adaptive concept, OE requirements and the control flow of the trading negotiation protocol. Section 3 describes the application and the architecture models and specification data needed by on-line metrics. Section 4 presents the efficiency and the resource matching metric and concludes with a numerical application of these metrics showing how they can be used by the OE. Section 5 concludes the paper.

[1]European IST-FET Project 027611

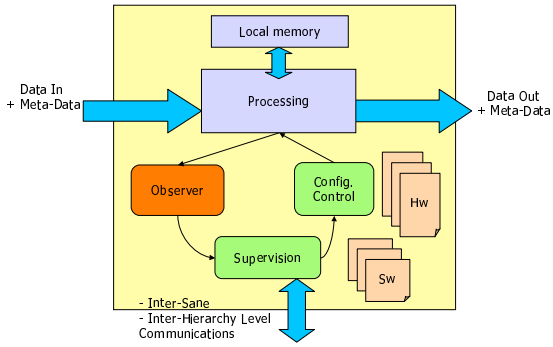

Figure 1: The SANE concept

2 SANE OE concepts

2.1 Separation of adaptation concerns

Self-adaptation may occur at all levels of application control. At the lowest level of the control flow we consider a sequence of non-conditional operations that can be modeled as a data-flow graph to be mapped onto a reconfigurable functional unit. At a higher level the OE partitions application tasks (processes) onto a set of distributed reconfigurable processors SPs according to soft or hard real-time constraints. Auto-adaptive application can be specified as a set of tasks including a configuration task that can create, kill and modify other tasks according to application specific metrics and adaptation rules. The same Self Adaptive Network Entity (SANE) concept can be used to specify self-adaptation on all levels.

The Æther SANE concept depicted in Fig.1, is applicable at all levels of hierarchy, so the approach can be implemented in different ways. The figure shows a loop composed of four blocks. The processing part of the configurable architecture can be a gate, a co-processor, a processor or a multiprocessor SOC. The observer, feeds the controller with measurements such as temperature, data-rate, etc needed during the decision process. The controller selects the right configurations and configures the architecture. Finally, the communication interface transports control signals to/from other SANE or to upper control layers that require information for global decisions. For instance the ID of current selected configuration can be provided.

1-4244-1517-9/07/$25.00 (C)2007 IEEE

2.2 Requirements

The OE must deliver four main services. The first service is the *dynamic and abstract specification* of computing, memory and communication resources. OE must be able to discover and qualify existing and/or configurable resources. This service requires specific effort to formalize application characteristics and architecture reconfiguration space. Another issue is the identification and certification of ambient processors for security concerns.

The second service includes *real-time, QoS management and monitoring* for qualifying applications distributed over dynamic platforms. There are two issues addressed in this service. The first one is the static hard or soft Real-Time analysis used before mapping decision. The second one is performed on-line and deals with monitoring. It is used to check whether applications performance objectives are met or not and so it stimulates task delegations for example.

The third service is the *trading of resources and services* between the SP network. This service is based on previous ones and decides task allocation, hardware / software partitioning and reconfiguration decisions. The service provides a trade-off between solution optimality and the cost of partitioning in terms of delay and power overhead. The OE must be able to fulfill application requirements in terms of resource allocation.

The last service deals with *task synchronization and communication protocols.*

2.3 OE partitioning strategy

2.3.1 Decision flow

The operating environment is implemented on each SANE processor. A SP will match the state of its resources with application needs when an application is launched. If there are sufficient resources locally, the SP will execute the application. If resources are insufficient the SP will negotiate with other SPs in the environment to find other resources as shown in Fig.2 . The negotiation includes

- *Request:* the initiator SP broadcasts the description of the application (task graph skeleton + objectives).
- *Bidding:* each available SP matches the request with the current state of its resources, relative to the specified constraints on QoS, energy efficiency, etc. Then each SP returns a bidding proposal containing resource characteristics (type and quantity) that it is willing to invest.
- *Partitioning:* based on received bids, proposals are evaluated and sorted. A selection relies on a credit-based calculation and a real-time analysis.
- *Contracting:* once above steps are completed, a partitioning solution (if successful) is identified and a contract is sent to selected SPs. Then, the SPs coordinating the execution of the application update their credit accounts.
- *Monitoring and adaptation:* adaptation implies dynamic resource allocation and (perhaps partial) redeployment of available resources to applications. An adaptor service checks monitoring data (e.g. communication bandwidth, task execution time, etc.) and compares it with specified thresholds. If contract violation occurs, a delegation procedure involving part of the application may by initiated. The same model is used for the whole application.

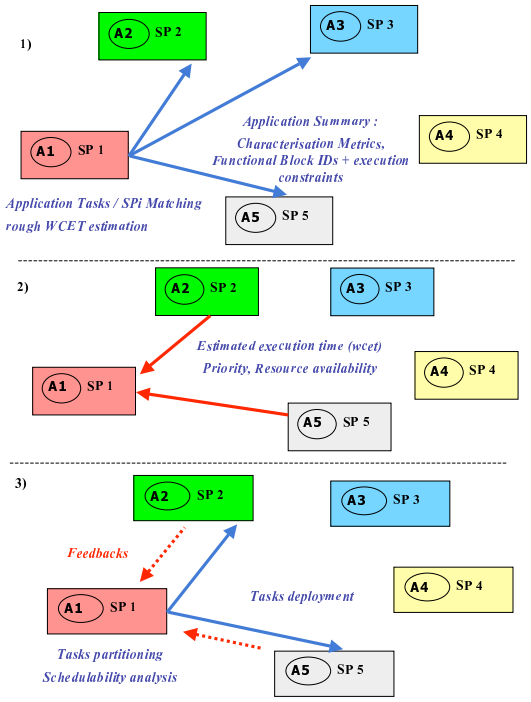

Figure 2: SANE Environment

2.3.2 Bidding

The selection of partner SPs must be adaptive, fast and low cost. For this reason, a two-step procedure has been chosen. The first step is selecting candidates based on a credit/debt protocol. This step is inspired by the evolution and adaptation processes of a famous contract net protocol [2]. Usually multi-agent systems are used in a bidding framework based on a monetary approach. In order to balance task mapping over the most efficient set of SPs, we use the concept of social debt within a cooperative system. The idea is the following: If OE A runs Task T for OE B (initiator), OE A increases its credit account with respect to B, and B's OE increases its debt regarding SP A.

A credit balance is used during the selection step of the mapping strategy. Possible contractors are sorted according to a cost function based on metrics and credits which favor efficiency and balance processing load. This policy has two positive effects. First, in order to obtain resources from other OEs, an OE will favor choices that reduce its debt in answer to proposals. Secondly, it will favor a choice that minimizes the impact of its contracts by accepting tasks for which it is efficient with certain priority. Thus an OE will try to send bids in order to balance its social debt with a minimum of effort.

The second step of the selection procedure is applied on a reduced set of SP candidates. A fast real-time analysis is performed to address hard real-time, soft real-time or best effort applications[3].

1-4244-1517-9/07/$25.00 (C)2007 IEEE

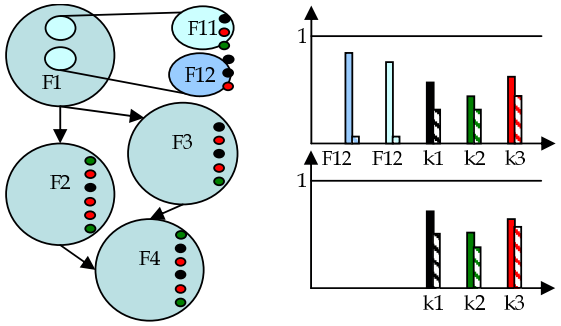

Figure 3: Application operations and energy signatures

3 System models

3.1 Application specification

Ambient computing implies a range of products that require standards and interoperability. This observation leads to the two following implementation assumptions. First, each device implements a Sane Virtual Processor (SVP) that can execute (or interpret) SANE applications more or less efficiently. Second, multimedia and telecom applications are implemented with standard functions which may have different specific HW/SW configurations.

Thus, an application can be specified as a dynamic task graph in which each task is a hierarchical control data-flow graph as shown in Fig.3. Leaf nodes in the graph correspond to basic instructions of the SVP. As a part of the semantics of the specification model, some meta-data are introduced during compilation across all hierarchy levels to guide metrics calculation for partitioning and scheduling. Meta-data includes:

- F_i_ID:standard function ID potentially available on target architectures.
- N: number of calls i.e. function instances.

3.2 Architecture model

A SP can be interpreted as an abstraction of a processor with a reconfigurable and multi-granular instruction set. A SP is composed of basic SANEs that can collaborate to create a SVP or a coarse grain function. A basic SANE can be a reconfigurable functional unit or a processor core depending on implementation.

The instruction set is mapped onto SANEs that can be more or less specialized for executing different classes of computation. At an abstraction layer above the SVP, each SP can provide a set of coarse-grain instructions implementing specific functions (F_i_ID). The configuration granularity may vary depending on the degree of specialization of the target architectures. For instance, in the case of 4G telecom applications, the function could be a *Log-Map*, a SISO decoder or a complete turbo-decoder. Characteristics (power, execution time) of coarse-grain instructions depend on SANE allocations. The characteristics are provided as estimates based on linear functions of the number of SANEs.

Finally, the architecture specification can be adapted in two ways. First, function characteristics can be updated according to observations. Second, in the ambient computing context, configurations for new functions (e.g. partial bit-streams) can be downloaded from a network data-base using a coarse grained instruction cache policy, in a way that is similar to firmware updates in current portable devices.

4 Metrics

4.1 Motivations

Metrics have already been used in the domain of co-design for selecting architectural models according to application characteristics [4], [5]. However, these metrics are used off-line because they requires a lot of computing resources; they are defined for specific architectures such as GPP, DSP or FPGA, and thus cannot be reused for on-line matching with any type of architecture. Two types of metrics are defined: energy efficiency metrics ($\mathcal{E}_H^A$) and resource matching metric(U). Efficiency metrics estimate the efficiency of an architecture for an application based on the architecture specialization of certain functions and categories of instructions while the resource matching metrics determine the possibility of application execution on an architecture and depict the architectural free and used amount of computing resources during application execution.

4.2 Efficiency metrics

The goal of the efficiency metrics is to quantify the suitability of application execution on architectures; this allows the use of policies like "architectures execute applications for which they are specialized". This approach differs from minimizing total application energy consumption and so, low power architectures are not favored. The overall goal is not to minimize the total energy needed to execute an application as for portable systems but rather to select architectures that are efficient in application execution. Efficiency and total energy optimization do not necessarily have the same goals.

Consider for instance a low power portable DSP architecture ($H1$: for hardware number one) and a legacy DSP ($H2$). The two may satisfy application objectives (performance needs), but the legacy DSP may be more efficient than the low power portable DSP although the latter consumes less. Efficiency maximization will favor $H2$, while total energy minimization will favor $H1$

4.2.1 Efficiency Metric Definition

The same energy efficiency metric is used to match hardware (H) energy efficiency to execute different size of operations. An operation could be an application, a function (coarse grain instructions) or an individual instruction. All operations are noted here by A for simplicity. The efficiency metric $\mathcal{E}_H^A$ is the proportion of energy E_P^{AH} used by A's operations on hardware H divided by the total amount of energy E_T^{AH} consumed while executing A on H. The proper amount of energy E_P^{AH} is the energy used by the hardware operator that executes the operation A during one execution. While the total amount of energy E_T^{AH} includes in addition to that: 1) the energy used to decode application instructions in processor case (or to configure a dedicated accelerator) in order to execute the operation on the hardware operator and 2) The

1-4244-1517-9/07/$25.00 (C)2007 IEEE

memory access needed by the operation. $\mathcal{E}_H^A$ must be between zero and one. If $\mathcal{E}_H^A$ is close to one the architecture H is better suited to execute A.

$$\mathcal{E}_H^A = E_P^{AH} / E_T^{AH} \quad (1)$$

4.2.2 Architecture Parameters

The relationship between application and architecture can be resumed as the application executes its instructions on architecture. An application can compose an architecture's basic instructions into a function. This function can be implemented as a coarse grain instruction in an accelerator on another architecture. Thus, instructions have different performance and energy consumption on different hardware. This concept motivates variable instruction set and variable granularity architectures.

We categorize all instructions that an architecture can execute. Instructions in each category k have nearly the same performance and energy weights. For example we group $+, -,$ max and min in the same instruction category. We denote the set of all instruction categories by $\mathcal{C}$. It is assumed that all instructions of category k have almost the same amount of energy E_P^{kH} and E_T^{kH}, even-though, in the absolute, this assumption may not fit exactly real-world implementation. Instruction that execute on a processor may have different execution time and consume different energy depending on whether its arguments are in a cache or not, even for basic operators such as addition for instance, the consumed energy depends on the instruction operands values. However this kind of abstraction is correct for two reasons: 1) a high level of abstraction is required for system level light weight decision and 2) the system is intrinsically adaptive which means that metrics can be dynamically updated. So, from now on we denote by E_P^{kH} and E_T^{kH} the mean value $\overline{E}_P^{kH}$ and the mean total energy $\overline{E}_T^{kH}$ consumed by instructions of category k.

Architecture signature

The architectural efficiency achieved by executing an instruction of category k is noted by $\mathcal{E}_H^k$. It is the ratio of energy consumed by the executed instruction E_P^k and the overall energy consumed while executing this instruction E_T^k. The signature of an architecture is defined by the graph of $\mathcal{E}_H^k$ over the set of all instruction categories $\mathcal{C}$. Fig.3 shows the energy signature of two architectures.

$$\mathcal{E}_H^k = E_P^{kH} / E_T^{kH} \quad (2)$$

Let's define another architectural parameter R_H^k that indicates the weight ratio of an instruction or the energy it consumes relative to other instructions.

$$R_H^k = E_T^k / E_R \quad (3)$$

Here E_R is a reference consumed energy that can be defined as $\sum_{k \in C} E_T^{kH}$, $\overline{E}_T$, $\min(E_T^k)$ or a constant energy value. In our case $E_R = \min(E_T^k)$. There is no need to communicate this parameter since $\mathcal{E}_H^A$ is computed by the SP that knows its parameters.

4.2.3 Application parameters

In order to calculate $\mathcal{E}_H^A$ for application A's instructions, we assume that A has N_A^k instructions of category k. Thus, (1) can be written as:

$$\mathcal{E}_H^A = \frac{E_P^{AH}}{E_T^{AH}} = \frac{\sum_{k \in C} N_A^k \, E_P^{kH}}{\sum_{k \in C} N_A^k \, E_T^{kH}}$$

$\mathcal{E}_H^A$ can be represented as a function of application parameter N_A^k and architectural parameters $\mathcal{E}_H^k$ and R_H^k as:

$$\mathcal{E}_H^A = \frac{E_P^A}{E_T^A} = \frac{\sum_{k \in C} N_A^k \, R_H^k \, \mathcal{E}_H^k}{\sum_{k \in C} N_A^k \, R_H^k}$$

Global algorithm

Since architecture H may implement certain functions that application A uses, the histogram N_A^k depends on the architecture H where the application executes. Fig.3 shows the energy signature and the histogram of an application on two different architectures, the first one implements two coarse grain function in addition to basic instructions. The application specification section defines the application as a hierarchical task graph. If the function k exists in the architecture then that level of the hierarchy can be used by considering the function meta-data N as N_A^k. If not, it is necessary to enter the function definition to reach instructions known by the architecture H. In the worst case these could be basic instructions.

The extension of instruction efficiency matching metrics to take into consideration coarse grain instructions requires a traversal of the application description with a depth first recursive algorithm. This algorithm uses application and architecture description models:

Function $[\mathcal{E}_H^A, R_H^A]$ $calcMatch$(App A, Archi H)
$MN \Leftarrow 0$
$MD \Leftarrow 0$
Set children = A.getChlidren() //get the directly first level children
For each element F in children
If F $\in H$ // F is implemented as an instruction
// read $\mathcal{E}_H^F$ and R_H^F from architecture specification
Else
$[\mathcal{E}_H^F, R_H^F] \Leftarrow calcMatch(F, H)$
End if
$MN \Leftarrow MN + \mathcal{E}_H^F \, R_H^F \, F.N$
$MD \Leftarrow MD + R_H^F \, F.N$
End for
M = MN/MD
R = MD
Return [M, R]
End function

4.3 Performance Metrics

Today's increasingly complex applications require more and more computing resources. In the following section

1-4244-1517-9/07/$25.00 (C)2007 IEEE

we will qualify computing resource needs of one category of instructions applications and generalize the concept for multi-categories applications. Generalization for any type of application will be addressed in future work. The defined performance metric allows us to online match an application's objectives with an architecture's capabilities.

4.3.1 Application Specification

For clarity we consider the following assumptions:1) all instructions are of the same temporal weight, so any instruction can execute on the architecture in one cycle. 2) the architecture is characterized by it's clock speed ν_H i.e. the number of cycles that the architecture executes every second and p_H the number of instructions that it executes every cycle. 3) application is characterized by static execution graph generated form its dynamic graph and the application objective states that it should executes in not more than d_A seconds. The total number of instructions in the application is N_A and the critical path length is λ_A. We characterize the application by the following parameters:

- $\overline{p}_A$, $\widehat{p}_A$: respectively, the mean and maximum application parallelism considering execution on an infinitely big architecture. The application will execute in a time (or number of cycles) equivalent to the length of its critical path. So, $\overline{p}_A$ (resp. $\widehat{p}_A$) is the mean (resp. the maximum) number of instructions per cycle that can execute during application execution along its critical path. So, $\overline{p}_A = N_A/\lambda_A$. One could argue that $\overline{p}_A$ and $\widehat{p}_A$ cannot be calculated exactly since determining the optimal number of processors that execute the application on its critical path is an NP-complex problem due to application dependency. But the values can be estimated off-line using fast ALAP scheduling for instance. $\overline{p}_A$ and $\widehat{p}_A$ characterize the dependency of application operations statistically.

- $\overline{P}_A, \widehat{P}_A$: respectively, the least acceptable mean and maximum application parallelism: $\overline{P}_A$ (resp. $\widehat{P}_A$) is the mean (resp. the maximum) number of executed instructions during the life time of the application per time unit (seconds) considering the slowest application execution that meets performance objectives.
  So, $\overline{P}_A = N_A/d_A$ and $\widehat{P}_A = N_A\,\widehat{p}_A/\overline{p}_A\,d_A$. $\widehat{P}_A$ is calculated by increasing the number of instructions to $\widehat{p}_A$ for each cycle of application execution. This is the same as assuming an application having $\widehat{p}_A\,\lambda_A = \widehat{p}_A\,N_A/\overline{P}_A$ instructions.

Notice that $\overline{p}_A$ and $\widehat{p}_A$ are measured in instructions per cycle while $\overline{P}_A$ and $\widehat{P}_A$ are measured in instructions per second.

If the architecture instruction set can take advantage of all architectural parallelism the maximum number of instructions that the architecture can execute during one second is $P_H = p_H\,\nu_H$.

Metric Definition

We define the mean and the minimum unused ratio of the architecture while executing the application as $\overline{U}_H^{d_A}$ and $\check{U}_H^{d_A}$ respectively. The architecture can execute the application when this ratio is positive and the ratio indicates the unused or the still available percentage of resources. Otherwise, the ratio is negative and the architecture capabilities are smaller than application requirements. The value of the ratio in this case indicates the percentage of the resources that we still have to acquire relative to the total application required resources.

$$\overline{U}_H^{d_A} = \frac{P_H - \overline{P}_A}{\max(P_H; \overline{P}_A)} = \frac{P_H - N_A/d_A}{\max(P_H; N_A/d_A)} \tag{4}$$

$$\check{U}_H^{d_A} = \frac{P_H - \widehat{P}_A}{\max(P_H; \widehat{P}_A)} = \frac{P_H - N_A\,\widehat{p}_A/\overline{p}_A\,d_A}{\max(P_H; N_A\,\widehat{p}_A/\overline{p}_A\,d_A)} \tag{5}$$

$\overline{U}_H^{d_A}$ and $\check{U}_H^{d_A}$ are between -1 and 1. We say that the architecture is able to satisfy the application objectives in average if $\overline{U}_H^{d_A}$ is positive and we can prove that it will satisfy them if $\check{U}_H^{d_A}$ is positive. In fact $\check{U}_H^{d_A}$ corresponds to an upper bound of application instruction number. This corresponds to a maximum $\hat{p}_A$ for every application cycle.

It can be said that the architecture is infinitely big relative to an application if it is capable of executing it within λ_A cycles the critical path length. This condition will happen when $p_H \geq \widehat{p}_A$.

The above definitions correspond to the most relaxed execution of the application respecting application objectives i.e. d_A. An acceptable execution of the application T_{AH} should be smaller than d_A and bigger than $\overline{T}^m_{AH}$ (or bigger than $\widehat{T}^m_{AH}$ if viewed pessimistically). Where $\overline{T}^m_{AH}$ ($\widehat{T}^m_{AH}$) is the minimum possible execution time of application A on architecture H considering its mean value characteristics (resp. pessimistic scenario). $\overline{T}^m_{AH}$ and $\widehat{T}^m_{AH}$ correspond to maximum possible resource utilization during application execution. This maximum is 100% if the architecture is not infinitely bigger than the application. They are given by:

$$\overline{T}^m_{AH} = N_A/p_H\,\nu_H \tag{6}$$

$$\widehat{T}^m_{AH} = N_A\,\widehat{p}_A/\overline{p}_A p_H\,\nu_H \tag{7}$$

For any application of acceptable execution time T_{AH} the unused ratio of the architecture H while executing A is:

$$\overline{U}_H^{T_{AH}} = \frac{P_H - \overline{P}_A}{\max(P_H; \overline{P}_A)} = \frac{P_H - N_A/T_{AH}}{\max(P_H; N_A/T_{AH})} \tag{8}$$

$$\check{U}_H^{T_{AH}} = \frac{P_H - \widehat{P}_A}{\max(P_H; \widehat{P}_A)} = \frac{P_H - N_A\,\widehat{p}_A/\overline{p}_A\,T_{AH}}{\max(P_H; N_A\,\widehat{p}_A/\overline{p}_A\,T_{AH})} \tag{9}$$

An application execution profile can be defined as a couplet $(T_{AH}; \overline{U}_H^{T_{AH}})$ (or $(T_{AH}; \check{U}_H^{T_{AH}})$). $(T_{AH}; \overline{U}_H^{T_{AH}})$ (resp. $(T_{AH}; \check{U}_H^{T_{AH}})$) varies from $(\overline{T}^m_{AH}; \overline{U}_H^{\overline{T}^m_{AH}})$ to $(d_A; \overline{U}_H^{d_A})$ (resp. from $(\widehat{T}^m_{AH}; \check{U}_H^{\widehat{T}^m_{AH}})$) to $(d_A; \check{U}_H^{d_A})$)

These metrics allow the OE to evaluate whether an application or a part of the application can be executed on the architecture. They restrict the solution domain. The selection of final solution depends on other OE objectives social dept for instance.

1-4244-1517-9/07/$25.00 (C)2007 IEEE

$\overline{U}_H^{T_{AH}}$ can be used for soft real-time application matching while $\check{U}_H^{T_{AH}}$ can be used for hard real-time applications matching.

4.3.2 Categories of Instructions

It is possible to distinguish k categories of instructions. Architecture H can execute an instruction I^k of type k in T_H^k cycles. It has p_H^k accelerators of type k that run at ν_H^k cycles per second. So, the number of operations of type k that the architecture can execute in one second is $p_H^k\,[\frac{\nu_H^k}{T_H^k}]$

The integer value $[\nu_H^k/T_H^k]$ can be approximated by ν_H^k/T_H^k

So, the architecture H can approximately execute $p_H^k\,\nu_H^k/T_H^k$ operations of type k per second.

Application A has N_A^k instructions of type k that should be executed before d_A^k. $\overline{p}_A^k$, $\widehat{p}_A^k$, $\overline{P}_A^k$, $\widehat{P}_A^k$ the equivalent applicative parameters as defined above for operations of type k. We define the unused amount of resources of type k as:

$$\overline{U}_{kH}^{d_A^k} = \frac{P_H^k - \overline{P}_A^k}{\max(P_H^k;\overline{P}_A^k)} = \frac{p_H^k\ \nu_H^k/T_H^k - N_A^k/d_A^k}{\max(p_H^k\ \nu_H^k/T_H^k - N_A^k/d_A^k)} \qquad (10)$$

$$\check{U}_{kH}^{d_A^k} = \frac{P_H^k - \widehat{P}_A^k}{\max(P_H^k;\widehat{P}_A^k)} = \frac{p_H^k\,\nu_H^k/T_H^k - N_A^k\,\widehat{p}_A^k/\overline{p}_A^k\,d_A^k}{\max(p_H^k\,\nu_H^k; N_A^k\,\widehat{p}_A^k/\overline{p}_A^k\,d_A^k)} \qquad (11)$$

If $U_{kH}^{d_A^k}$ is smaller than zero we should consider partitioning the application.

4.4 Illustrative Example of Metrics

To illustrate on-line application architecture matching with our metrics, consider the execution of a multimedia codec application on: H_1, a basic GPP like processor, H_2, a PC with DSP capabilities (like OMAP), H_3, a PC with DSP and DCT accelerator (like OMAP_2), H_4, an efficient DSP (like Tensilica processor) and $H5$ an ASIC that directly implements the codec application.

The application and architecture characteristics along with the matching results for H_1, H_2, H_3, H_4 are shown in Table 1. The first column describes the codec on H_1, H_2 and H_4. The codec has a different histogram on H_3 because it has a different instruction set (implements a DCT).The application deadline d_A is the application period. Following raws describe instruction properties. H_1 runs at the same frequency as H_3 which is ten times smaller than the frequency of H_2 and H_4. H_2, H_3 and H_4 have the same architectural parallelism for basic instructions while H_3 has an accelerator for DCT. The last line in the table concerns H_3 only. H_2 and H_3 have the same efficiency $\mathcal{E}^k$, while H_4 is more efficient for major codec instructions. We consider the same E_T^k for all architectures.

H_1 would not meet the objectives of lot of the instruction set categories so we cannot consider executing the application on it. H_2, H_3 and H_4 all have positive $\overline{U}_k^{d_A^k}$, thus they can execute the application. Notice that H_3 meets the application objective and runs at the same frequency as $H1$ because it has a DCT accelerator. H_3 has a larger performance metric (unused amount of resources) than H_4 for DSP operations even though it runs at a lower frequency because a big part of the DSP operations are performed in the DCT core. This is not the case for the ALU. H_5 performance metric would be slightly larger than zero since it is an ASIC which is specific for this application. All resources will be used when executing the application.

	APP		H_1	H_2		H_3		H_4		$H_2H_3H_4$	H_1	H_2H_4	H_3
	$H_1H_2H_4$	H_3	p_H^k	p_H^k	$\mathcal{E}_H^k$	p_H^k	$\mathcal{E}_H^k$	p_H^k	$\mathcal{E}_H^k$	E_T^{kH}	$\overline{U}_k^{d_A^k}$		
Freq			10^7	10^8		10^8		10^7					
Period	0,041	0,041											
DSP32	3894400	12800	1	2	0,3	2	0,3	2	0,6	100	-0,89	0,53	0,98
ALU32	3640640	1233280	1	4	0,2	4	0,2	4	0,5	100	1,00	0,78	0,25
Bit	640	1280	1	6	0,04	6	0,04	6	0,04	1	-0,11	1,00	1,00
STR	460480	473920	1	32	0,001	32	0,001	32	0,001	1	-0,05	1,00	0,96
CJMP	430720	849920	1	8	0,5	8	0,5	8	0,5	1	-0,57	0,99	0,74
Loop	963520	520000	1	8	0,5	8	0,5	8	0,5	1	-0,88	0,97	0,84
M Acc	3549760	1147520	1	4	0,45	4	0,45	4	0,6	1000	-0,61	0,78	0,30
@	1059840	1031681	1	4	0,5	4	0,5	4	0,5	1	1,00	0,94	0,37
DCT H_3		21600				2	0,9			10000			0,97

Table 1: Application Characteristics

The Efficiency metric would not be calculated for $H1$ due to incompatibility. It gives 0.42 for H_2, 0.49 for H_3 and 0.59 for H_4. The efficiency of $H5$ would be above 0.85. If the crediting system is used, H_5 will be the most loaned in the group since it will often ask other SPs to execute its applications (other than the codec application). So, the OE will decide to execute the codec application on H_5.

5 Conclusion

The efficiency and performance metrics presented in this paper allows an operating environment (OE) to dynamically determine system capacities relative to real-time applications objectives. The performance metric matches spacial and temporal computing resources capacities with application objectives, while the efficiency metric favors the execution of application on specialized architectures. We generalized these metrics to tune along the application hierarchy. A credit based negotiation protocol was also introduced, it enables OE to partition the application over SPs in the environment. Future publications will present already defined equivalent memory and communication metrics and validation results from the SystemC simulator under development.

References

[1] The Real-Time Specification for Java Copyright 1994-2007 Sun Microsystems Inc, "http://java.sun.com/docs/books/realtime," .

[2] R.G.Smith, "The contract net protocol : High level communication and control in a distributed problem solver," IEEE *Trans. on computers*, vol. c-29, no. 12, pp. 1104–1113, dec 1980.

[3] M.Auguin F.Muhammad, F.Muller, "Contentions-conscious dynamic but deterministic scheduling of computational and communication tasks," in *ACM Symp. on Applied Computing (SAC)*, Dijon, France, 2006.

[4] Y.Le Moullec, N.Ben Amor, J-Ph.Diguet, J-L.Philippe, and M.Abid, "Multi-granularity Metrics For The Era Of Strongly Personalized SOCs," in *DATE*, Munich, Germany, Mar. 2003.

[5] D.Sciuto, F.Salice, L.Pomante, and W.Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *10th Int. Symp. on* H/S *Codesign*, Estes Park, USA, May 2002.

1-4244-1517-9/07/$25.00 (C)2007 IEEE