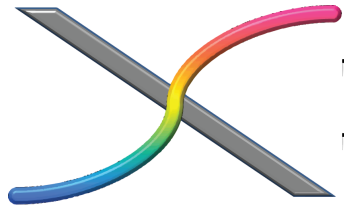


EXCESS

XPDL: Extensible Platform Description Language to Support Energy Modeling and Optimization

Christoph Kessler, Lu Li, Aras Atalar and Alin Dobre
christoph.kessler@liu.se, lu.li@liu.se

This project is part of the portfolio of the
A.3 – Advanced Computing and Complex System Unit
Communications Networks, Content and Technology DG
European Commission

www.excess.eu

Contract Number: 611183
Total Cost [€]: 3.31 million
Starting Date: 2013-09-01
Duration: 36 months

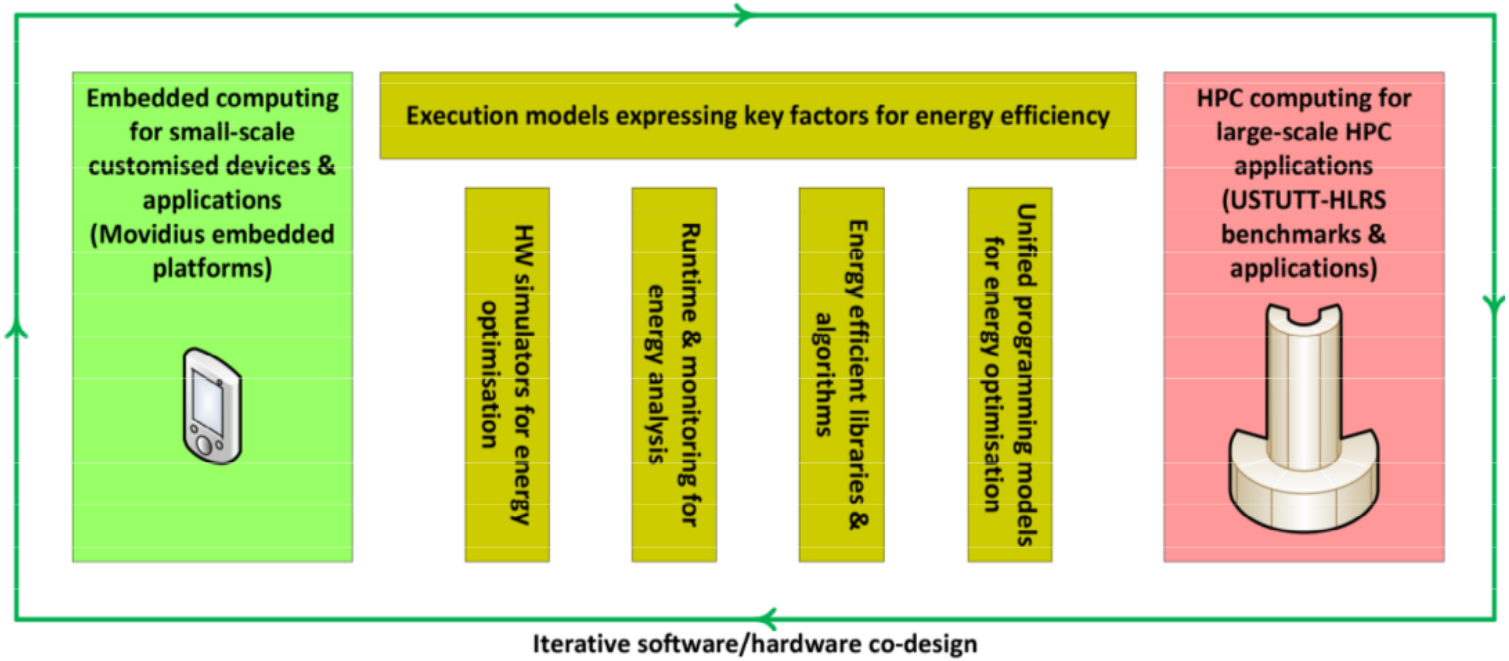
Agenda

- Motivation
- A Review of PDL
- XPDL Features
- XPDL Toolchain and Runtime Query API
- Related Work and Comparison
- Conclusion and Future Work

Acknowledgment

EXCESS Project (2013-2016)

- EU FP7 project
- Holistic energy optimization
- Embedded and HPC systems.
- More info: `http://excess-project.eu/`



Motivation

- Optimization:
  - Platform-independent: algorithmic improvement
  - Platform-dependent: SIMD, GPU etc
- Platform dependent optimization such as SIMD can yield significant performance and energy improvements.
- Usually platform dependent optimizations are manually tuned or partly automated.
  - Requires understanding of the machine-specific features.
- Automation of systematic platform-dependent optimizations is both interesting and challenging.
  - Adaptivity
- Retargetability
- **Prerequisite: a formal platform description language modeling optimization-relevant platform properties.**

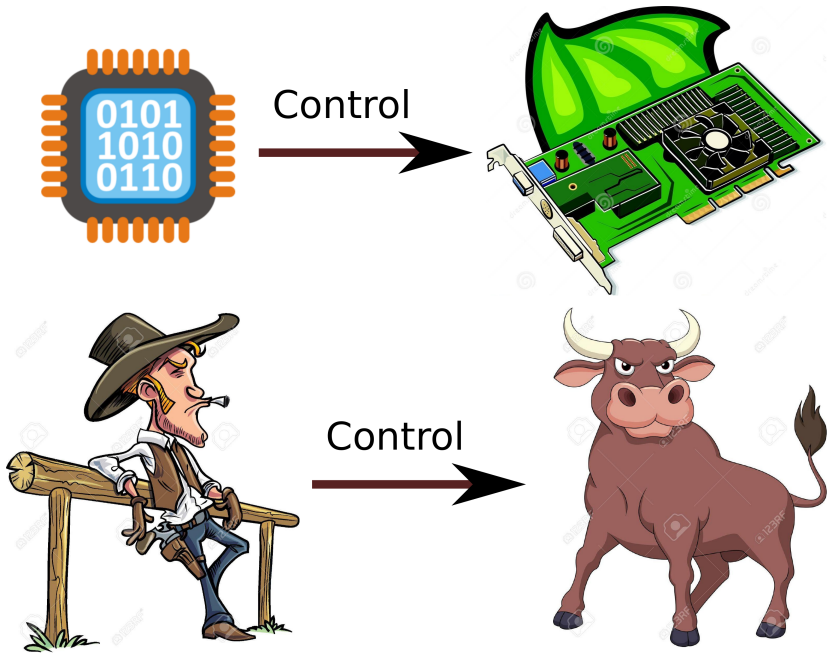
Motivation

Platform = Hardware + System Software

- Previous work: PDL (Platform Description Language)
  - Limitations: flexibility and scalability.
- XPDL is designed to overcome those limitations,
- Furthermore, new features are added to better support energy optimization, such as microbenchmark generation.

PDL: the Predecessor of XPDL

- PDL: PEPPHER Platform Description Language. (Sandrieser et al. ’12) [1]
- Developed in EU FP7 project: PEPPHER (2010-2012)
- XML-based
- First description language for heterogeneous systems
- Models the control structure of master and slave PUs.
- Enable conditional composition. (Dastgeer et al. ’14) [2]



A Review of PDL

- Main structuring by control relation (a software aspect!) rather than hardware organization
  - hard to change.
- Modularity issues.

```
<!-- XML HEADER -->
<Master id="0" quantity="1" putype="cpu" logicgroup="set01">
    <PUDescriptor>
        <Property fixed="true" xsi:type="cscPropertyType">
            <name>ARCHITECTURE</name>
            <value>x86</value>
        </Property>
        <!-- Additional Properties -->
    </PUDescriptor>
    <Worker id="1" quantity="1" putype="gpu">
        <PUDescriptor>
            <Property fixed="true" xsi:type="cscPropertyType">
                <name>ARCHITECTURE</name>
                <value>gpu</value>
            </Property>
            <!-- Additional Properties -->
        </PUDescriptor>
    </Worker>
    <Interconnect type="onesided" from="0" to="1">
        <!-- ICDescriptor -->
    </Interconnect>
</Master>
```

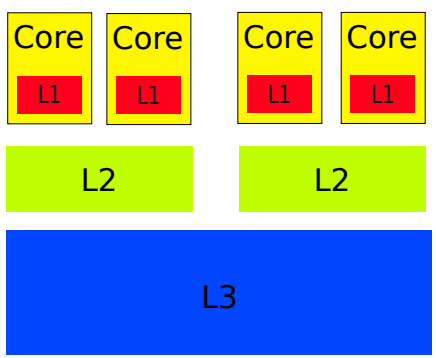
Figure 1 : A PDL example (Sandrieser et al.) [1]

XPDL Language Features

- Modular and extensible
- Control relation decoupled from hardware
- Syntax choice: XML
  - Mature tool support
  - Syntactic flavor does not restrict its applicability.
- System software modeling
- Power modeling
- Microbenchmark generation
  - For statically-unknown parameter values
- Standardized XPDL runtime query API
  - Available to application and tool chain:
  - E.g. libraries, compilers, runtime systems etc.

A Hello World XPDL Example

- Examples are only to illustrate XPDL language constructs, not meant to be complete.



(a) A Typical CPU Structure

```
1  <cpu name="Intel_Xeon_E5_2630L">
2    <group prefix="core_group" quantity= "2"  >
3      <group prefix="core" quantity= "2"  >
4        <!-- Embedded definition -->
5        < core   frequency="2" frequency_unit="GHz" >
6        <cache name= "L1"   size="32" unit="KiB" />
7        </core>
8      </group>
9      <cache name= "L2"   size="256" unit="KiB" />
10   </group>
11   <cache name= "L3"   size="15" unit="MiB" />
12   <power_model type="power_model_E5_2630L" />
13 </cpu>
```

(b) The corresponding XPDL code

Modular and extensible

- name: defining a meta-model, stores type information
- id: defining a model, stores object information

Figure 2 : Type reference and inheritance

Control Relation Decoupled From Hardware

```
1  < Master    id="0" quantity="1">
2     <peppher:PEDescriptor>
3        <peppher:Property fixed="true">
4        <name>ARCHITECTURE</name>
5        <value> x86 </value> ...
6     <peppher: Worker    quantity="1" id="1">
7      <peppher:PEDescriptor>
8        <peppher:Property fixed="true">
9        <name>ARCHITECTURE</name>
10       <value> gpu </value> ...
11    </peppher:Worker>
12 </Master>
```

Listing 1 : PDL example description for x86-core (Master) and gpu (Worker)

```
1  <system id="liu-gpu-server">
2    <socket>
3      < cpu    id="gpu-host" type= "Intel-Xeon-E5-2630L"  />
4    </socket>
5    < device    id="gpu1" type= "Nvidia-K20c"    />
6    <interconnects>
7      <interconnect id="connection1" type="pcie3" head="gpu-host" tail="gpu1" />
8    </interconnects>
9  </system>
```

Listing 2 : XPDL example description for such a GPU server

System Software Modeling

```
1  <system id="XScluster">
2   <  cluster  >
3    <group prefix="n" quantity="4">
4     <  node  >
5      <group id="cpu1">
6       <socket> <cpu id="PE0" type="Intel-Xeon-..." /> </socket>
7      </group>
8      <group prefix="main-mem" quantity="4"> <memory type="DDR3-4G" /> </group>
9      <device id="gpu1" type="Nvidia-K20c" />
10     <interconnects>
11      <interconnect id="conn1" type="pcie3" head="cpu1" tail="gpu1" />
12     </interconnects>
13    </node>
14   </group>
15   <interconnects>
16    <interconnect id="conn3" type="infiniband1" head="n1" tail="n2" />
17   </interconnects>
18  </cluster>
19  <software>
20   <hostOS id="linux1" type="Linux-..." />
21   <  installed    type="CUDA-6.0" path="/ext/local/cuda6.0/" />
22   <  installed    type="CUBLAS-..." path="..." />
23   <  installed    type="StarPU-1.0" path="..." />
24  </software>
25  <properties>
26   <property name="ExternalPowerMeter" type="..." command="myscript.sh" />
27  </properties>
28 </system>
```

Listing 3 : Example of a concrete cluster machine

Power Modeling

- A power model in XPDL consists of
  - power domains and their power state machines
  - microbenchmarks with deployment information

Modeling Power Domains

- Power domains: hardware components that must change state together.
- E.g. in Movidius Myriad2, each SHAVE core form a separate power island.

Modeling Power Domains

```
<power_domains name="Myriad1-power-domains">
  <!-- this island is the main island -->
  <!-- and cannot be turned off       -->
  <power_domain name="main-pd" enableSwitchOff="false">
    <core type="Leon" />
  </power_domain>
  <group name="Shave-pds" quantity="8">
    < power_domain   name= "Shave-pd"  >
      <core type= "Myriad1-Shave"   />
    </power_domain>
  </group>
  <!-- this island can only be turned off      -->
  <!-- if all the Shave cores are switched off -->
  <power_domain name="CMX-pd"
                switchoffCondition  = "Shave-pds off"  >
    <memory type="CMX" />
  </power_domain>
</power_domains>
```

Listing 4 : Example meta-model for power domains of Movidius Myriad1

Modeling Power State Machine

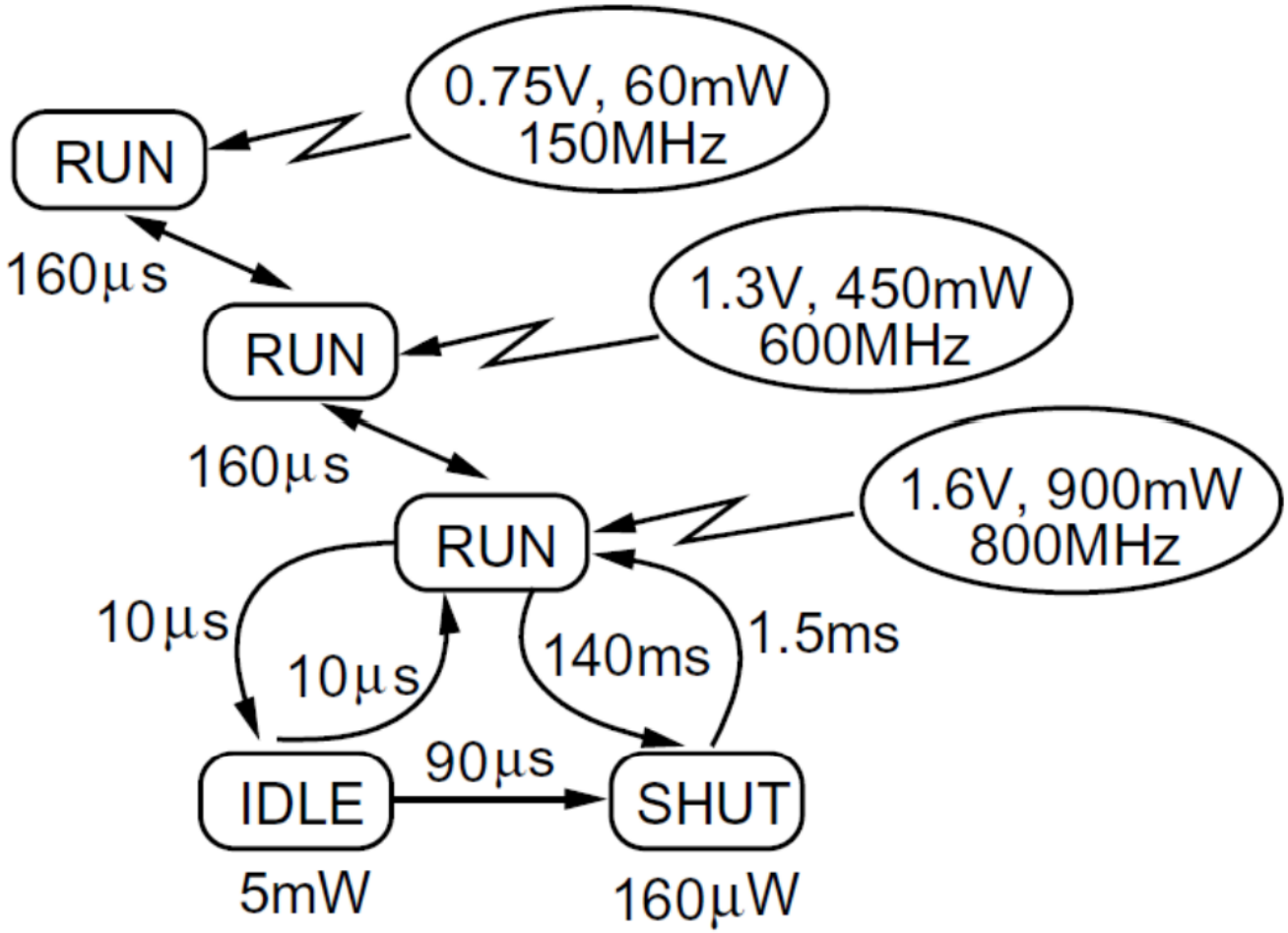


Figure 3 : Intel Xscale processor (2000) with voltage scaling and shutdown.
Source: Alexandru Andrei, PhD thesis 2007

Modeling Power State Machines

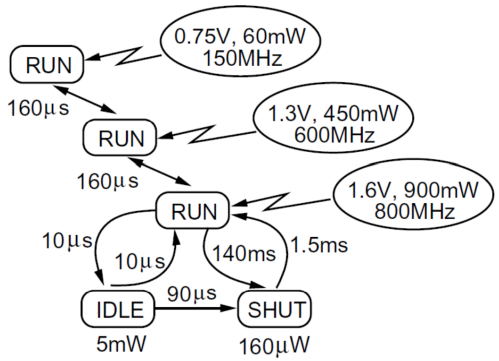


Figure 4 : Intel Xscale processor (2000)

```
1  <power_state_machine name="power-state-machine1"
2        power_domain  ="Intel-Xscale-pd" >
3    <power_states>
4      < power_state   name= "P1"     frequency  ="150" frequency_unit="MHz"
5   power="60" power_unit="mW" voltage="0.75" voltage="V" />
6      <power_state name="P2" frequency="600" frequency_unit="MHz"
7   power="450" power_unit="mW" voltage="1.3" voltage="V" />
8      ...
9    </power_states>
10   <transitions>
11    < transition   head= "P1"   tail= "P2"     time  ="160" time_unit="us" />
12    <transition head="P2" tail="P1" time="160" time_unit="us" />
13    ...
14   </transitions>
15 </power_state_machine>
```

Listing 5 : Example meta-model for a power state machine of Intel Xscale processor (2000)

Modeling Microbenchmarks With Deployment Information

```
<instructions name="x86-base-isa"
        mb  = "mb-x86-base-1"    >
 <  inst    name= "fmul"
         energy  = "?"    energy_unit="pJ" mb="fm1"/>
 <inst name="fadd"
        energy="?" energy_unit="pJ" mb="fa1"/>
 <  inst    name= "divsd"  >
   <data frequency="2.8"
 energy= "18.625"    energy_unit="nJ"/>
   <data frequency="2.9"
 energy="19.573" energy_unit="nJ"/>
   <data frequency="3.4"
 energy="21.023" energy_unit="nJ"/>
 </inst>
</instructions>
```

Listing 6 : Example meta-model for instruction energy cost

```
<microbenchmarks id= "mb-x86-base-1"
 instruction_set="x86-base-isa"
 path="/usr/local/micr/src"
   command  ="mbscript.sh">
 <microbenchmark id="fa1" type= "fadd"    file= "fadd.c"
   cflags   ="-O0" lflags="..." />
 <microbenchmark id="mo1" type="mov" file="mov.c"
 cflags="-O0" lflags="..." />
</microbenchmarks>
```

Listing 7 : An example model for instruction energy cost

XPDL Toolchain and Microbenchmark Generation

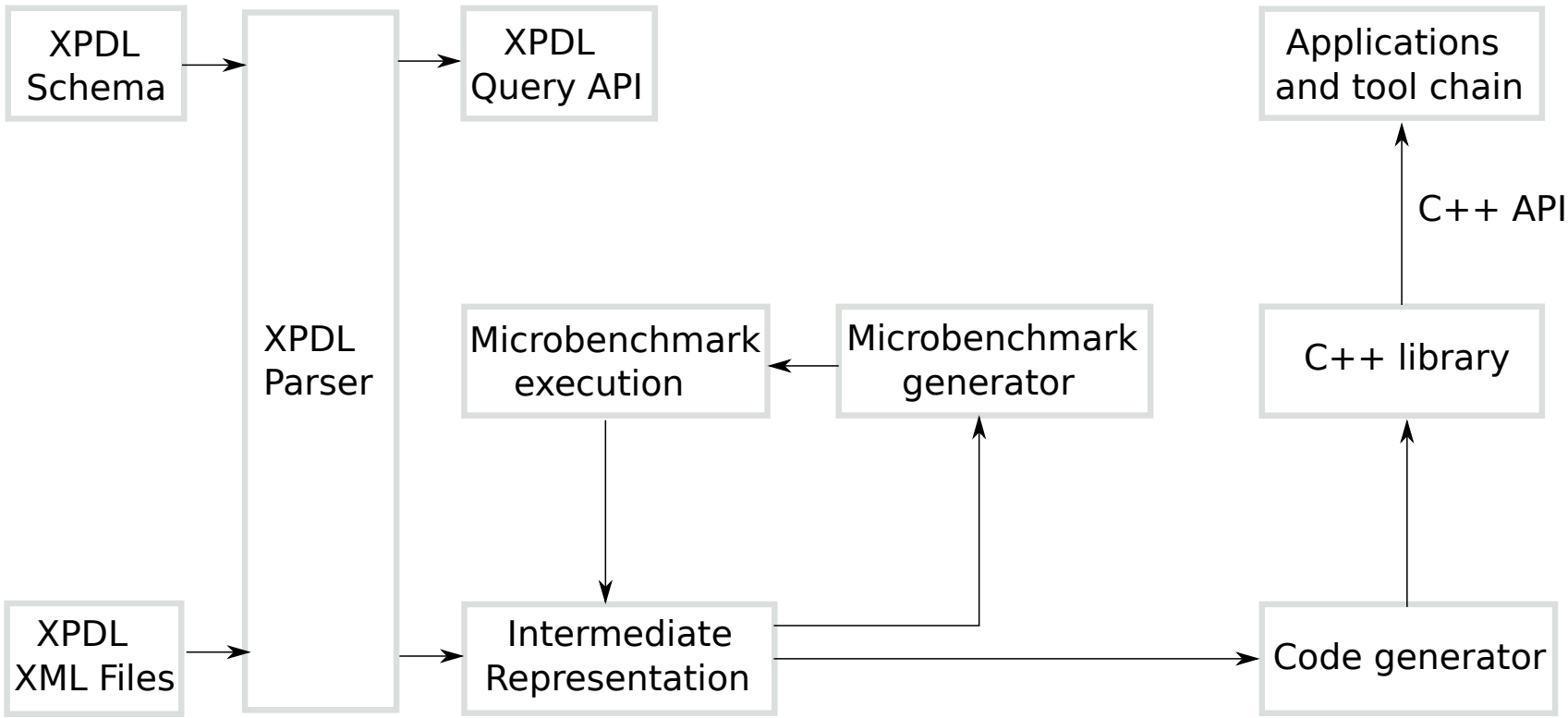


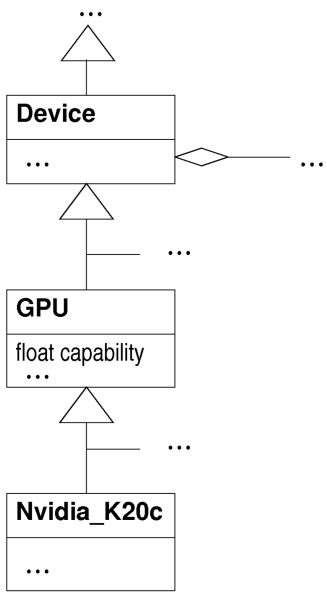
Figure 5 : XPDL Tool Chain Diagram

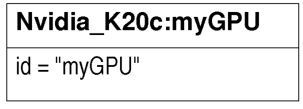
XPDL Runtime Query API

- Initialization of the XPDL run-time query library
- Functions for browsing the model tree
- Functions for looking up attributes of model elements
- Model analysis functions for derived attributes
  - Static inference, e.g. PCIe bandwidth
  - Aggregate numbers,
    e.g. get the total static power
    or the total number of GPUs

Different Views on XPDL

	UML	XML	C++

Metamodel level		`<device type = "gpu" ... >`
		`...`
		`</device>`
		
		`<gpu type = "Nvidia_K20c"`
		`capability="2.0"... >`
		`...`
		`</gpu>`
			`class device:ProcessingUnit`
			`{`
			`...`
			`}`
			
			`class GPU:device`
			`{`
			`float capability;`
			`...`
			`}`
			
			`class Nvidia_K20c:GPU`
			`{`
			`capability=2.0`
			`...`
			`}`

Concrete model level		`<gpu type="Nvidia_K20c"`
		`id="myGPU" ...>`
		`...`
		`</gpu>`
			`Nvidia_K20c myGPU`
			`= new Nvidia_K20c("myGPU");`

Related Work and Comparisons

- PDL
  - XML-based
  - Modeling control structure
  - cf. XPDL: control relation decoupled, modular etc
- Hwloc:
  - detects and represents the hardware resources visible to the machine's operating system
  - cf. XPDL: not limited to OS
- HPP-DL
  - In EU FP7 REPARA project
  - Purpose: to support static and dynamic scheduling of software kernels to heterogeneous platforms
  - JSON syntax
  - cf. XPDL: modeling of power states, dynamic energy costs, system software, distributed specifications, runtime model access or automatic microbenchmarking.

Summary of Contributions

- We propose XPDL, a portable and extensible platform description language
- Modular and extensible
- Software roles decoupled from hardware structure
- System software modeled
- Microbenchmark generation
- Open source tool chain on the way,
  soon at `http://www.ida.liu.se/labs/pelab/xpdl/`

Future work

- Continue to finish the XPDL toolchain
- Demonstrate applicability for adaptive energy optimization.
- Generate runtime library that exposes API for dynamic hardware resource management.
- Demonstrate applicability for retargeting static optimizers and source code generators.
- Write automatic converters of vendor-specificADLs (e.g. hwloc) to XPDL

Bibliography I

M. Sandrieser, S. Benkner, and S. Pllana, “Using explicit platform descriptions to support programming of heterogeneous many-core systems,” *Parallel Computing*, vol. 38, no. 1-2, pp. 52–65, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819111001396

U. Dastgeer and C. W. Kessler, “Conditional component composition for GPU-based systems,” in *Proc. MULTIPROG-2014 workshop at HiPEAC-2014 conference, Vienna, Austria*, Jan. 2014.