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
Agenda

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

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

(a) A Typical CPU Structure

(b) The corresponding XPDL code
Modular and extensible

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

```xml
<device name="Nvidia_K20c" extends="Nvidia_Kepler">
  <compute_capability>3.5</compute_capability>
  <param name="num_MM" value="13"/>
  <param name="coresperSM" value="192"/>
  <param name="cfrq" frequency="706" unit="MHz"/>
  <param name="gmsz" size="5" unit="GB"/>
  ...
</device>

<device name="Nvidia_Kepler" extends="Nvidia_GPU">
  <compute_capability>3.0</compute_capability>
  <const name="shmtotalsize" size="64" unit="KB"/>
  <param name="Llsize" configurable="true" type="msize" range="16, 32, 64" unit="KB"/>
  <param name="shmsize" configurable="true" type="msize" range="16, 32, 64" unit="KB"/>
  <param name="num_SM" type="integer"/>
  <param name="coresperSM" type="integer"/>
  <param name="cfrq" type="frequency"/>
  <param name="gmsz" type="msize"/>
  <constraints>
    <constraint expr="Llsize + shmsize == shmtotalsize"/>
  </constraints>
  <group name="SMs" quantity="num_SM">
    <group name="SM">
      <group quantity="coresperSM">
        <core type="..." frequency="cfrq"/>
      </group>
      <cache name="L1" size="Llsize"/>
      <memory name="shm" size="shmsize"/>
    </group>
  </group>
  <memory name="global" size="gmsz"/>
  ...
  <programming_model type="cuda6.0,...,opencl"/>
</device>
```

Figure 2: Type reference and inheritance
Control Relation Decoupled From Hardware

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

Listing 2: XPDL example description for such a GPU server
Listing 3: Example of a concrete cluster machine

```
<system id="XScluster">
  <cluster>
    <group prefix="n" quantity="4">
      <node>
        <group id="cpu1">
          <socket>
            <cpu id="PE0" type="Intel–Xeon—..." />
          </socket>
        </group>
        <group prefix="main–mem" quantity="4">
          <memory type="DDR3–4G" />
        </group>
        <device id="gpu1" type="Nvidia–K20c" />
        <interconnects>
          <interconnect id="conn1" type="pcie3" head="cpu1" tail="gpu1" />
        </interconnects>
      </node>
    </group>
    <interconnects>
      <interconnect id="conn3" type="infiniband1" head="n1" tail="n2" />
    </interconnects>
  </cluster>
  <software>
    <hostOS id="linux1" type="Linux—..." />
    <installed type="CUDA–6.0" path="/ext/local/cuda6.0/" />
    <installed type="CUBLAS—..." path="..." />
    <installed type="StarPU–1.0" path="..." />
  </software>
  <properties>
    <property name="ExternalPowerMeter" type="..." command="myscript.sh" />
  </properties>
</system>
```
A power model in XPDL consists of
- power domains and their power state machines
- microbenchmarks with deployment information
Power domains: hardware components that must change state together.

E.g. in Movidius Myriad2, each SHAPE core form a separate power island.
Listing 4: Example meta-model for power domains of Movidius Myriad1

```xml
<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">
    <memory type="CMX" />
  </power_domain>
</power_domains>
```
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)

Listing 5: Example meta-model for a power state machine of Intel Xscale processor (2000)
Modeling Microbenchmarks With Deployment Information

Listing 6: Example meta-model for instruction energy cost

Listing 7: An example model for instruction energy cost
XPDL Toolchain and Microbenchmark Generation

Figure 5: XPDL Tool Chain Diagram
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**

```
Device
  ...
  ...  ...
```

**XML**

```
<device type = "gpu" ... >
  ...
</device>
```

```
<gpu type = "Nvidia_K20c"
  capability="2.0"... >
  ...
</gpu>
```

**C++**

```cpp
class device:ProcessingUnit
{
  ...
}
```

```cpp
class GPU:device
{
  float capability;
  ...
}
```

```cpp
class Nvidia_K20c:GPU
{
  capability=2.0
  ...
}
```

```
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-specific ADLs (e.g. hwloc) to XPDL