Improving Programmability of Heterogeneous Many-Core Systems via Explicit Platform Descriptions

Martin Sandrieser, Siegfried Benkner, Sabri Pllana

Research Group Scientific Computing
Faculty of Computer Science
University of Vienna, Austria
http://www.par.univie.ac.at/
Talk Outline

- Motivation & Context
- Explicit Platform Descriptions
- Case Study & Experiments
- Related Work
- Conclusions & Future Work
Heterogeneous MC Architectures

Move towards heterogeneous many-core architectures

• Better performance/power ratio
• Different types of cores; same cores but different clock frequencies
• Specialized cores for specific tasks/application domains
→ Parallelization & Specialization (mitigate Amdahl’s law)

Examples

• Cell Processor: PPU + 8 SPUs
• SARC Research Processor
• CPU + GPU/Accelerators
• Tianhe-1A, Roadrunner, TSUBAME
• Nvidia Tegra, AMD Fusion, SandyBridge
Programming Heterogeneous Many-Cores

Much harder than for homogeneous systems

- Explicit memory management
  (DMA transfers, double buffering, local stores ...)
- Partitioning of code for different cores
- Different memory models, ISAs, compilers, APIs, programming models

Variety of different HW/SW properties

- Implicitly defined by execution models, run-time libraries, OS, ...
- Different users may have different degree of platform specific knowledge

How can platform specific information be managed by users & tools?
Explicit Platform Descriptions

Goal: Make platform specific information explicit and available in a systematic way to tools and users.

XML-based platform description language (PDL)

- Capture different aspects of heterogeneous platforms
  - **Control views**: delegation of computational tasks between processing units; hierarchical organization of PUs
  - **Hardware / Software properties** (e.g., core-count, memory sizes, available libraries)
- Supports expression of **platform usage patterns** (e.g. Master-Worker)
- Not a hardware description language!
  
  Programmer centric view on available resources (→ platform)
Platform Descriptors

**Processing Units** (PUs)
- Master (initiates program execution)
- Worker (executes delegated tasks)
- Hybrid (master & worker)

**Memory Regions**
- Express key characteristics of memory hierarchy
- Can be defined for all processing units

**Interconnects**
- describe communication facilities between PUs

**Properties**
- Hardware and software properties using generic key/value mechanism
Platform Descriptors – XML Schema
Examples

GPGPU System

Cell B.E. System
EU Project PEPPHER

Performance Portability & Programmability for Heterogeneous Many-Core Architectures

- EU ICT Call 4, Computing Systems; Start: Jan. 2010, 36 Months;
- Coordinated by University of Vienna; http://www.peppher.eu

Goal: Enable portable, productive and efficient programming of heterogeneous many-core systems.

Holistic Approach

- Higher-Level Component-Based Support for Parallel Program Development
- Auto-tuned Algorithms & Data Structures
- Compilation Strategies
- Runtime Systems
- Hardware Mechanisms

Crosscutting Application Domains: Embedded – General Purpose – HPC
PEPPHER Goal

Methodology & framework for development of performance portable code.

- Execute same application efficiently on different heterogeneous architectures.
- Support multiple parallel APIs: **OpenMP, OpenCL, CUDA**, pThreads, ...

**Approach**

- **Component-based** approach (multiple implementation variants of tasks)
- **Performance-aware tasks**
- **Intelligent runtime system** to select best task variant for given platform
A PEPPHER Example Application

Cholesky factorization

FOR \( k = 0..TILES-1 \)

\[
\text{POTRF}(A[k][k])
\]

FOR \( m = k+1..TILES-1 \)

\[
\text{TRSM}(A[k][k], A[m][k])
\]

FOR \( n = k+1..TILES-1 \)

\[
\text{SYRK}(A[n][k], A[n][n])
\]

FOR \( m = n+1..TILES-1 \)

\[
\text{GEMM}(A[m][k], A[n][k], A[m][n])
\]

Utilize expert written components:
BLAS kernels from MAGMA and PLASMA

Implementation variants:
• multi-core CPU (PLASMA)
• GPU (MAGMA)

Make into PEPPHER component:
Interface, implementation variants + performance meta-data
**PEPPHER Approach**

### Mainstream Programmer
- Component-based application with annotations

### Expert Programmer (Compiler/Autotuner)
- Component impl. variants for different cores, algorithms, inputs ...  

### Compiler
- Intermediate task-based representation

### Runtime System
- Dynamic selection of "best" implementation variant

### Target Platforms
- Feed-back of measured performance

---

**Programmer**
- Identify performance critical parts
- Make into **performance-aware components**
- Provide implementation variants for different core architectures or utilize expert components

**PEPPHER framework**
- Management of components and implementation variants
- Compilation/code generation
- Component variant selection
- Dynamic, performance-aware task scheduling
PDL Usage Scenario

Mainstream programmer
- Builds application using components
- Annotates component calls possibly referencing PDL descriptor
- High-Level coordination mechanisms

Expert programmer
- Implements component variants for specific architectures
- Provides component meta-data
  - platform requirements,
  - performance model, etc.

using PDL descriptors
Case Study – Source-to-Source Translator

Prototype based on ROSE framework

Input:
- Sequential C + task annotations + distribution of array parameters
- Implementation variants (CPU, GPU) from task repository
- Tasks:
  - No side-effects, only local references
  - Communicate via parameters only

Output:
- C + calls to PDL-specific runtime system
- Annotated tasks split into smaller tasks based on distribution (if any)
- Pre-selection of task variants (PDL)
- Registration of tasks with runtime system (StarPU)
Case Study – Source-to-Source Translator

Execute Annotation

- Task name + Execution group (reference to PU-group attribute in PDL)
  + Parameter data distributions
Case Study – Source-to-Source Translator

- Testmachine:
  8 CPU cores and 2 GPUs
  (Nvidia GTX480, GTX285)

- Application with DGEMM task:
  Blas variant & CuBlas variant

- Three different PDL configurations

- Generate **StarPU runtime** calls with help of PDL

**StarPU**: University of Bordeaux; http://runtime.bordeaux.inria.fr/StarPU/
Related Work

Sequoia (Stanford)
- Tree abstraction of memory modules
- Provides portability via external target descriptors

Hierarchical Place Trees (Rice University)
- Tree abstraction of memory hierarchy
- Locality-based dynamic scheduling

Hwloc (University of Bordeaux)
- Portable API for hardware locality information (used in StarPU)
- Focus on low-level topological HW information (NUMA nodes, sockets, caches)

Task Offloading
- HMPP (CAPS, France), StarSS (UPC, Barcelona), Offload (Codeplay, UK), PGI, ...

Algorithmic Choice
- Elastic Computing (Univ. Florida)
- PetaBricks (MIT)
Conclusion and Future Work

Platform Descriptors

• Hierarchical abstraction of heterogeneous MC systems (control view of PUs)
• Used at different levels of software stack
• Make platform-specific implementation explicit (e.g. task implementation variants)
• Can help with code generation for heterogeneous platforms

Future Work

• Use of PDL within PEPPHER framework (performance portability)
• Performance modeling of platform-specific task variants
• Use of PDL within runtime system
• Use of PDL-like mechanisms in auto-tuners