Hardware-aware Computing on Multicore Processors and Accelerators

Rainer Buchty

Universität Karlsruhe (TH) – Forschungsuniversität
Institut für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur
Introduction

The case for multicore processors
Introduction

Moore’s Law

- Still valid:
  Transistor count per area doubles every 18-24 months
  - Passed 1 billion in 2006
  - Tukwila: 2 billion in 2008 (delayed)

*Moore’s Extrapolation 1965*  
*Intel Architectures 1970-2005 ©Intel*
Introduction (cont’d)

Hitting the complexity wall

- Increasing technological problems limiting complexity of uniprocessors
  - Conventional speed-up has come to an end
  - Power issues (leakage, switching)
  - Cache size / pipeline stage vs. clock speed

- Besides: What else can we do?
  - Microarchitectural approaches exhausted (ILP, BP, $, …)

- What remains: coarse-grained parallelism
  - Example: Improved resource use supporting TLP (HT)

Exploiting coarse-grained parallelism

- Utilize Moore’s law
- Circumvent complexity wall
Introduction (cont’d)

What future brings...

- Further exploitation of Moore’s law therefore means **system-level integration**
  - Exploit coarse-grained parallelism → Manycores
  - System on Chip (SoC)

- Result: **Supercomputers on a chip**
  - Massively parallel architectures
  - GPUs: massively parallel (240-800 cores)
  - Intel Polaris: 1 TFLOP on a chip
  - IBM Cell: “Supercomputer on a chip”

- Near future: **Use of dynamically reconfigurable hardware**
Introduction (cont’d)

**Technology scaling**

<table>
<thead>
<tr>
<th>Arising Problems</th>
<th>Different solution strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication issues</td>
<td>Hierarchical architectures</td>
</tr>
<tr>
<td>Memory access and cache sharing</td>
<td>P2P connections (HT, CSI/QP)</td>
</tr>
<tr>
<td>Programmability and programming model</td>
<td>HW support for programming models (TM, CUDA)</td>
</tr>
</tbody>
</table>

Two basic approaches leading to different architectures

1. **Scaling of (homogeneous) computing clusters**
   - “Cluster on a chip”
   - Homogeneous multi-/manycore architectures

2. **Dedicated architecture integration/shrinking**
   - Mix of general-purpose and dedicated cores
   - Heterogeneous multi-/manycore architectures
Multicore Architectures

Homogeneous Multicores

- Scaling of SMP method
  - Processors of equal capabilities
  - Exploitation of coarse-grained parallelism (Task/Thread level)
  - Often found: shared memory
    (Local L1, Shared L2 and higher)
- Comparatively easy to develop and implement
  - However: memory bandwidth issues
  - Migration to P2P communication instead of shared buses

Picture Source: Intel
Multicore Architectures (cont’d)

Heterogeneous Multicores

- Scaling of ASMP method
  - Different specialization of cores
  - Master/Slave Concept
  - Often found: communication-based approach

- “True” Heterogeneity
  - Mix of different individual cores

- Same ISA Heterogeneity
  - Same ISA
  - Different hardware configuration (speed, cache, microarchitecture)

Picture Source: Intel, Nvidia, Xilinx
First Multicore Dilemma: Speed and Parallelism

- **Clock speed**
  - Due to power consumption and heat dissipation, speed per core reduced
  - Single-core performance reduced
  - Reaching the peak performance requires full use of all cores

- **Parallelism**
  - Coarse-grained parallelism (task level) achieves improved overall workload balancing and responsiveness
  - Single-threaded applications don’t harness multicore power
  - Application need to be specifically programmed
Multicore Architectures (cont’d)

Second Multicore Dilemma: Memory Performance

- Processor speed $\gg$ Memory Speed
  - System speed is dominated by memory performance: Memory Wall
  - Installation of a cache memory hierarchy
  - However: memory wall persists, you just hit it later

- Trend increased with multicore architectures
  - Multiple cores, one memory
  - Bus transport capacity
  - Central system buses $\rightarrow$ P2P approach

*Pictures from Wulf, McKee: Hitting the Memory Wall, CS-94-48, Dec. 1994*
Exploiting Parallelism

- To exploit parallelism, programs need to be specifically designed with respect to the **underlying hardware** or supported programming models.

- **Parallel programming models**
  - Data-parallel (Vectorization)
  - Explicit communication, message-based (MPI)
  - Implicit communication, i.e. shared memory
    - Pessimistic synchronization (OpenMP)
    - Optimistic synchronization (TM)

- **Suitability** for multi- and manycore architectures?
  - Not just scaled down supercomputers!
  - Aspects of heterogeneity and dynamics
  - → Hardware-aware programming
Target Architectures

...at a glance
Multicore Architectures

Architecture Approaches

- **Scaling of homogeneous uniprocessors**
  - Thread-/Task-level parallelism
  - Multiplication of general-purpose cores
  - Same ISA and (typically) same speed
  - Example: IA32 multicores

- **FP computation accelerators**
  - Data parallelism – “Sea of FP units”
  - Symmetric, grid-like arrangement
  - Hierarchy: Host/Master, Accel./Slave
  - Host provides control flow
  - Examples: GPUs, ClearSpeed

- **Heterogeneous architectures**
**Data-parallel Architectures**

**Application defines architecture**
- **High degree of data parallelism** enables linear speed-up
- **Simple operations**, therefore low number of branches
- **High arithmetic intensity** masks memory latency
- **Example: GPUs**

![Diagram](attachment:diagram.png)

**General-purpose Architecture**

**Data-parallel Architecture**
Data-parallel Architectures (cont’d)

Rise of the GPUs

- Use of **GPU hardware** (Shaders) for computation
- Pixel Shader: **Single-precision FPU**
- Data-intensive applications show high degree of parallelism
- GPU offers plenty of shaders
- **Architectural support for GPGPU**
  - Memory hierarchy leveraging amount of data traffic
  - Massive SMT (several 1000 threads) supported by HW
  - Synchronization

Picture Source: Nvidia
Data-parallel Architectures (cont’d)

Example: Nvidia Tesla

- Up to 30 multiprocessors per device
- Per multiprocessor...
  - 8 processors
  - 16kB shared memory
  - 8kB constants/texture cache
- Per Processor
  - 8k/16k 32-Bit Registers
- Up to 2GB system memory
- Parameters defined by Compute Capability
Data-parallel Architectures (cont’d)

Example: AMD HD400

- Up to 10 SIMD engines offering up to 160 thread processors equaling 800 Stream Cores
- Thread processor represented as a 5-issue VLIW processor
  - Awaits operation for all stream cores per instruction
- Support for double precision
Merging Host and Accelerator

Example: Cell BE

- **PowerPC Processor Element** (PPE)
  - Control flow and housekeeping
  - Access to main memory

- **7/8 Synergistic Processor Unit** (SPU)
  - SIMD-style Vector Units (16x8, 8x16, 4x32, 2x64 bit operands)
  - 4x single precision or 2x double precision FP operation
  - 256kB local memory

- **Memory Flow Controller** (MFC)
  - Interface between SPU and Interconnect Bus
  - DMA data transfer

- **Element Interconnect Bus** (EIB)
  - Ring bus
  - Connects SPUs (via MFC) and PPE
Merging Host and Accelerator (cont’d)

Cell BE (cont’d)

![Diagram showing Cell BE components](Picture Source: Sony)
Dynamic Heterogeneity: FPGAs

- **FPGAs**\(^1\) are like **ASICs**\(^2\)
  - Integration of complex logic functions
  - Fine-granular use of logic resources
  - Design of domain-specific units

- **FPGAs** are like processors
  - User-programmable (configurable)
  - Standardized and tested product, no expensive chip set-up and testing costs

- **Today: Platform FPGAs**
  - Integration of dedicated functional units
  - RAM block integration
  - Support for fast communication
  - Integration of processor cores as dedicated hardware

---

1. Field-Programmable Gate Array
2. Application-Specific Integrated Circuit
Dedicated connectivity support

- Support for various standards such as OC-x, HyperTransport, Infiniband, PCI Express, Gigabit Ethernet
- Programmable on-chip terminators
- Programmable sampling delay (in steps of 80ps)
- Support for multiple clock domains (up to 24)
FPGAs: Example Virtex 4 (cont’d)

DSP support

- Direct support for 48-bit MAC operations
- 18x18-bit signed multiplier w/ sign-extension to 48 bits
- 48-bit adder/subtractor
- Rounding
- Dedicated, cascading buses for input/output data

Picture Source: Xilinx
Embedded Processing

- Integration of PPC405 hard macro
- Support for user-defined instruction-set extension
  - 8 user-defined instructions
  - Dedicated interface to processor pipeline (APU)
- Standardized bus interfaces (IBM CoreConnect)
The price to pay...

FPGAs are not for everyone

- FPGA system integration
  - In the distant past: dedicated coprocessor interface
  - In the not-so-distant past: peripheral buses (slow!)
  - Today: high-speed interfaces like HyperTransport and QPI

- Using FPGAs requires dealing with HDLs like VHDL and Verilog

- Alien concept to software programmers
  - Everything is concurrent unless explicitly serialized
  - Dealing with hardware issues (interfacing, timing, area)
  - Implementation of functionality on rather low level
Hardware Description Languages

Example: VHDL

- Standardized hardware description language (HDL), initial IEEE standard 1987
- VHDL = VHSIC HDL = Very High Speed Integrated Circuits HDL
- Models different parts of design cycle
  - Algorithmic specification (behavioral description)
  - Hardware-affine specification (register transfer level, RTL)
  - Simulation
- Ada-derived syntax enhanced by constructs for hardware design
- High level of abstraction compared to early hardware design languages, still higher abstraction level than e.g. Verilog
VHDL: Description Anatomy

**Interface Definition (entity)**
- Entity solely models input/output interface
- Per module only one entity allowed

**Function modeling (architecture)**
- Describes actual behavior
- Multiple architectures per model possible, e.g. behavioral vs. synthesizable, area vs. speed
- Support for generics and their configuration

**Test Bench**
- Functional simulation and checking of timing behavior
- Pre- and post-fit simulation
Modeling Sequentiality

- Assignments in VHDL always concurrent
- Sequential logic explicitly modeled using process construct
  - Process triggers to signal changes (sensitivity list)
  - Sequential algorithm describes desired logic
  - However: no time delay between process start and end
- No dedicated key word for storage-element synthesis
  - Modeling desired behavior by explicit description
  - Signal change (level/edge)
  - Storage element type (D, T, J/K, R/S)
Example architecture

Cray XD-1

- Originally designed by Octigabay
- Per blade: Dual-core AMD Opteron plus FPGA (Virtex 2/4) connected through HT-like interface
- Blades connected by Infiniband
- **Rudimentary software support** on OS level (FPGA configuration, start/stop execution)
- No dedicated integration of FPGA development software

Source: Cray, Inc.
Reconfigurable coprocessors

- Coprocessor concept, i.e. static general-purpose ISA plus coprocessor commands
- Reconfigurable coprocessor hardware
- Reconfigurable coprocessor commands
- Example: MOLEN, TU Delft
Heterogeneity and Dynamics (cont’d)

Reconfigurable system architectures

- **Fully dynamic infrastructure**
  - Dynamically changing processing elements
  - Dynamically changing interconnect using overlay networking

- **Example:** DodOrg, Univ. Karlsruhe (TH)

- **Use of Self-X capabilities** for managing system complexity

---

**Ziel−Funktionen**

- Automatische Planung
- Adaptive Komponenten
- Monitor

**System**

- Cells with common structure
- Multi−grained Interconnection
- Peripheral Devices

**System 1**

- FPGA Cell
- DSP Cell
- Monitor Cell
- I/O Cell

**System 2**

- FPGA Cell
- I/O Cell

**System...**

30/51 HPCS Tutorial: Hardware-aware Computing
Rainer Buchty • 21.06.2009
Multicores Architectures: Programming

Programming model mismatch

- Established parallel programming models show **coarse-grained division** into
  - Data parallelism
  - Message passing
  - Shared memory

- No appropriate support for **platform-specific requirements**
  - Hardware-aware programming required, e.g. IBM Cell
  - Use and configuration of accelerator hardware, dedicated hardware design (if required) and manual invocation
  - Dynamic behavior typically not covered, must be integrated into program logic
Multicore Architectures: Programming (cont.)

Programming and Execution Frameworks

- **Runtime environment** tightly coupled to the programming model
  - Communication channels
  - Synchronization methods
  - Parallelism control (fork/join)

- **Vendor-specific approaches**
  - Programming and optimization tool suite (IBM Cell)
  - Dedicated threading support (Nvidia CUDA)
  - Focus on heterogeneous execution (RapidMind, OpenCL)
  - Dynamic parallelization (Intel Ct)
  - Dynamic hardware/software migration (IBM Lime)
Parallel Programming

Programming heterogeneous architectures
Cell Broadband Engine (cont’d)

Program development

- Cell is, by design, a message-based architecture
- No dedicated language extension
- Basic service and application libraries
- Programming environment and supporting tools for debugging and performance analysis
- Standard compilers and object tools (GNU-based)
- Future: Serial programming model + Octapiler
The CUDA Platform

Overview

- Platform for program development and parallel processing on NVIDIA GPUs
- Library functions
- Shared source code (C/C++)
  - Compiler forks for CPU/GPU architectures
  - Preprocessor
- Hardware abstraction
  - GPU generation
  - Compute Capabilities
  - Install-time assembly

Diagram:

- CUDA Optimized Libraries: `math.h`, FFT, BLAS, ...
- Integrated CPU + GPU C Source Code
- Nvidia C Compiler
- Nvidia Assembly for Computing (PTX)
- CPU Host Code
- Standard C Compiler
- GPU
- CPU

HPCS Tutorial: Hardware-aware Computing
Rainer Buchty • 21.06.2009
CUDA: Thread Management

- GPUs are **Massively parallel architectures** (up to 240 cores)
- Exploiting thread-level parallelism
- GeForce 8: **12k concurrent threads**
- GeForce 9: **40k concurrent threads**

![CUDA Thread Management Diagram]
CUDA: Thread Management (cont’d)

**Programmer’s View**

- **Block** has up to 3 dimensions w/ individual threads of a block running on the same multiprocessor. Hardware restricts size to max. 512
- **Grid** has one or two dimensions. Max. $2^{16}$ elements per dimension
- **Defined by programmer** using extended C semantics
- Typically fixed block size but variable grid size

**Example:** $x <<< nblocks, 256 >>> (\ldots)$

- $x <<< nblocks$: grid dimension, number of grids per block
- $256 >>> (\ldots)$: number of threads (256) per block
**CUDA: Thread Management (cont’d)**

<table>
<thead>
<tr>
<th>Thread programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Easy programming model</strong></td>
</tr>
<tr>
<td>- No explicit thread management / thread programming by application programmer</td>
</tr>
<tr>
<td>- Parallel formulation of compute kernel</td>
</tr>
<tr>
<td><strong>Hardware support by dedicated thread manager</strong></td>
</tr>
<tr>
<td><strong>Transparent run-time management</strong></td>
</tr>
<tr>
<td><strong>Access blocking of data shared between threads</strong></td>
</tr>
<tr>
<td>(Threadlock elimination)</td>
</tr>
<tr>
<td>- Theoretically: no <strong>deadlocks</strong> possible</td>
</tr>
<tr>
<td><strong>Where required: explicit barrier synchronization using</strong></td>
</tr>
<tr>
<td><code>syncthreads</code></td>
</tr>
</tbody>
</table>
CUDA: Thread Management (cont’d)

Register use and decomposition

- Compiler determines optimal register allocation (Reg./Thread)
- **Example:** 12288 threads, 128 TPs (GeForce 8) → 10 registers/thread
- Realistic ballpark number: **20-32 registers/thread**
  - Limits number of threads to about 4000-6500 concurrent threads
CUDA: Programmer’s view

Finding a suitable decomposition

- CUDA eases threading but does not assist parallelization
  - Analysis and decomposition by the programmer
  - Splitting an application into individual problems
  - Data distribution and assignment
  - Code transformation

- Challenge: Finding a suitable decomposition
  - Goal: constantly high GPU use
  - Various influences
    - Compute Capabilities (# of cores, memory/register size)
    - Number of used register / size of local data sets
    - Maximum size of data shared among threads
CUDA: Programmer’s view (cont’d)

Programmer support

- Providing kernel routines
  - Basic Linear Algebra Subprogram (BLAS)
  - E.g.: SAXPY → \( y[] = a \times x[] + y[] \)

C/C++ extensions: Compiler Switches

- Locality/Access control
  - __global__: global access to kernel routine by entire application
  - __shared__: shared variable in local TP memory

C/C++ language extensions

- API calls for storage management and data access
  - Memory management: `cudaMalloc()`, `cudaFree()`
  - Data transfer: `cudaMemcpy()`, `cudaMemcpy2D()`
AMD StreamComputing

Similarities and Differences

- Concept similar to CUDA approach
  - Related syntax and nomenclature
  - **Kernel**: Computation to be performed on every input element
  - **Stream**: Manifold of same-type data elements on which the same operation is performed in parallel

- Based on Stanford’s BrookGPU (→ Brook+)

- Differences to Nvidia/CUDA mainly on HW level
### Open Compute Language (OpenCL)

- Conceived by Apple (WIPO Patent application handed in 2008)
- Further development steered by non-profit **Khronos Group**
- **Picks up idea of stream computing**
  - Division into control flow on host and application kernel to be executed on accelerator(s)
  - Not bound to specific accelerator vendor
  - Support for Cell BE and certain DSPs
- **Based on Low-Level Virtual Machine (LLVM)**
  - LLVM delivers virtual instruction set
  - Compilation and optimization on LLVM
  - Dynamic mapping to target hardware during runtime
OpenCL (cont’d)

Programming using OpenCL

- **Compute Device** dissects into Compute Units
- **Compute Unit** contains number of **Processing Elements**
- Application description similar to CUDA or Brook
  - Parallel formulation, not loop
  - Source code quite easily transformable by exchange of keywords

Source: Khronos Group
RapidMind Multicore Development Platform

RapidMind Framework

- Platform for **run-time distribution of workload** on a parallel, heterogeneous system
- Dissects into API, Front-end, & Back-end
- Core element: **RapidMind C/C++ API**
- No new toolflow required, but integrateable into existing projects and environments

Picture Source: RapidMind.com

HPCS Tutorial: Hardware-aware Computing
Rainer Buchty • 21.06.2009
RapidMind Front- and Backends

- **Code Optimizer**: analysis and optimizations of computations, overhead elimination
- **Load Balancer**: planning and synchronization for best possible resource use (processing cores)
- **Data Manager**: reducing data bottlenecks
- **Logging/Diagnostics**: bottleneck detecting and signalling
- **Processor support by dedicated back-ends**
  - IA32 processors (AMD, Intel)
  - GPUs: ATI/AMD and NVIDIA
  - Cell BE on Cell Blade, Cell Accelerator Board, PS3
  - Debugging back-end
Using RapidMind

What happens where?

1. Replacing numerical data types (FP, Integer) by corresponding RapidMind data types

2. Computation during runtime: identifying computation sequences, dynamic compilation into a RapidMind program object (PO)

3. Stream processing: guided parallel execution of POs on target hardware during runtime

Picture Source: RapidMind.com
Upcoming approaches

**Intel C for Throughput Computing (Ct)**

- **Thread creation driven by data structures** representing vectors, hierarchies, hash tables, etc.
- Support for heterogeneous task parallelism
- Language support and run-time system for flexible, parallel collection-oriented building blocks
- Ct provides parallel-ready versions of common data types
- Superset of C/C++

**IBM Liquid Metal (Lime)**

- **Breaking the barrier between HW and SW**
- Codemorphing not only into desired machine code, but also HW description / configuration bitstream
- Java-based approach
Literature
Links

- **ATI Stream Technology**  

- **Ct: C for Throughput Computing**  
  [http://techresearch.intel.com/articles/Tera-Scale/1514.htm](http://techresearch.intel.com/articles/Tera-Scale/1514.htm)

- **Liquid Metal (Lime)**  

- **Nvidia CUDA Zone**  

- **OpenCL Overview**  
  [http://www.khronos.org/opencl/](http://www.khronos.org/opencl/)

- **RapidMind Multi-core Development Platform**  
  [http://www.rapidmind.net](http://www.rapidmind.net)
Hardware-aware Computing on Multicore Processors and Accelerators

Rainer Buchty

Universität Karlsruhe (TH) – Forschungsuniversität
Institut für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur