Kernfragen:
Multicore-Prozessoren in der Industrie

InvasIC-Kolloquium, FAU Erlangen
10. Februar 2012

Urs Gleim
Siemens AG – Corporate Technology
System Architecture & Platforms
Parallel Processing Systems Program
urs.gleim@siemens.com
Technical Challenges

Vision
Strong Focus on Embedded Systems

Industrial control systems
- e.g. rolling mill

Transportation systems
- e.g. railway, car

Medical equipment
- e.g. magnetic resonance imaging

Communication systems
- e.g. network switch

Energy management
- e.g. smart meter
Vision
Future Evolution in General

Multicore processors will provide still increasing computing power, even in embedded environments with electric power limitations.

Progress in wireless communication allows for flexible interconnect topologies.

Increasing semiconductor integration levels provide powerful on-chip networking facilities.

Hardware and software commoditization and Open Source provide powerful platforms.

Past (Single-Core)

Today (Multi-Core)

Future (Many-Core)
Challenges

1. Consolidation

**Hardware Consolidation**
- reduced number of controllers and DSPs
- increased flexibility due to software solutions

**Real-time system architectures**
- ensure real-time behavior with partitioning and virtualization technologies

**Architectures for mixed critical systems**
- separation of safety-relevant subsystems on the same processor
- Efficient development and evolvability of safety-critical systems (e.g., independent certification of safety-critical (software) components)
- Dependability in open systems

---

Challenges

2. Decentralization

**Decentralization**
- e.g., HMI/SCADA and MES functionality integrated in controller devices

---
Challenges

3. Heterogeneity

Heterogeneous multi-/many-core architectures
- Utilization of special purpose cores of multicore processors (portable programming models, load balancing)

Hardware accelerators
- Optional acceleration units (e.g., GPUs)

Cloud computing
- Flexible deployment
- Scalability of resources
- Communication bottleneck
- Security and data privacy

Challenges

4. Security

The more complex and interconnected a system is, the bigger the number of security vulnerabilities: We have to defend against cyber-attacks.
5. Energy Management

Power Efficiency

- Mobility: Energy consumption of on-board electronics must be minimal. (public transportation, eCar)
- Mobile devices driven by battery or energy harvesting (e.g., healthcare in rural areas with unreliable energy supply)
- Limited installation space in industrial devices or energy management (waste heat problem)

→ Universal energy management architecture needed!

6. Programming Models

For parallel hardware architectures today’s Programming Models are
- too complex
- error prone
- not scaling with number of processing units
- non-deterministic

We need programming models that are
- suitable for the masses
- “taking parallelism mainstream” (Microsoft)
- development efficiency comparable to sequential software development
- abstraction from hardware architectures to the greatest possible extent
- compatibility to common programming languages (huge code base available)
- flat learning curve for developers
Challenges

7. Migration Strategies

Parallel processing units enable
- doing more → data volume increases constantly
- doing faster → interactive work with IT systems in healthcare and industry (simulation)

huge code base of sequential code to be parallelized
- where to start?
- how to parallelize?
- how to ensure correctness?

Summary

The 7 Challenges of Embedded Software Development

1. Consolidation
   - shift from HW to SW
   - utilization of multi-/many-core systems
   - taking into account safety and real-time requirements

2. Decentralization
   - flexible deployment of functionality in distributed systems

3. Heterogeneity
   - heterogeneous multi-/many-core architectures
   - hardware accelerators
   - cloud computing

4. Security
   - data privacy
   - protection against manipulation

5. Energy management
   - power-efficient hard- and software

6. Programming models
   - Development efficiency and future-proofness
   - Portability, HW-independence
   - Scalability with processing power (more cores)

7. Migration strategies
   - utilize parallel hardware preserving existing code bases
The Importance of Multicore Processors

Why Parallel Processing?
Power Wall / Frequency Wall

Energy density limits core frequency

Source: Fred Pollock, Intel, New Microprocessor Challenges in the Coming Generations of CMOS Technologies, Micro 02

Illustration: A. Tovey Source: D. Patterson (U.C. Berkeley)
Why Parallel Processing?

Moore’s Law

- Moore’s Law
  
  The number of transistors on an integrated circuit increases exponentially, doubling approximately every two years.

- Intel predicts this trend through 2029
  (IDF 2008)

- What does this mean for multi-core CPUs?
  
  200-2000 cores/CPU in mass market in 10 years!

---

Why Parallel Processing?

Rapidly Growing Number of Cores

Expected evolution of networking SoCs:
- Number of cores increases by 1.4 × / year
- Core frequency increases by 1.05 × / year

source: International Roadmap for Semiconductors 2009 (http://www.itrs.net/)

Reality Check (summer 2010)

- Standard x86 based 48 core Server: 6-12 cores/CPU, up to 4 CPUs
- Embedded CPUs: 2-4 cores
Why Parallel Processing?
How does Hardware Evolve?

Example Calxeda's EnergyCard
- 4 quad-core ARM Cortex-A9 cores

⇒ 16 cores


---

Why Parallel Processing?
How does Hardware Evolve?

HP's Redstone servers
- 18 of Calxeda's Energy Cards

⇒ 72 cores

Why Parallel Processing?

How does Hardware Evolve?

Rack HP’s Redstone servers
- 4 x 18 cards
- 288 cores
- up to 3000 chips in one rack
- 12,000 cores


Why Parallel Processing?

Trends

more CPU cores (SMP) more processors (SMP) integrated graphics and chipset features

SIMD instructions (ILP)

heterogeneous cores (AMP)
HW accelerators, DSPs
What can we do with Multicore-Processors?

Benefits of Parallel Architectures

- Performance
- Power
- Heat
- Bill of Material
Why Parallel Processing?
Chances by Multi-Core Architectures (1/3)

- Performance
  - more accuracy
  - increased throughput
  - decreased latency
  - additional computing power for innovative features

Future performance only possible by parallel software

Why Parallel Processing?
Chances by Multi-Core Architectures (2/3)

- Power / Heat
  - energy efficiency
  - healthcare in rural areas with no reliable power infrastructure
  - limited installation space for industry automation and energy devices (waste heat problem)
  - mobility: power consumption of board electronics to be kept minimal (public transportation as well as eCar)
Why Parallel Processing?
Chances by Multi-Core Architectures (3/3)

- BoM
  - lower production costs by replacing specialized processing HW by general purpose processors
  - use cheaper sensor technology and actuating elements and compensate quality loss by intelligent software
  - high performance low cost products

Our Work
Consequences for Software Development

The Free Lunch is Over*

* Herb Sutter; The free lunch is over: A fundamental turn toward concurrency in software”; Dr. Dobb’s Journal, 30(3), 2005

Today’s software!

Goal

Utilization of all core ➔ only possible with parallel software!
**Example 1: Software Design Patterns**

**What is a Design Pattern?**

A design pattern in software design is a general reusable solution to a commonly occurring problem within a given context.

- **Name**
- **Context**
- **Problem**
  - Forces
  - Requirements for the solution
- **Solution**
  - Structure
  - Dynamics
- **Consequences**
  - Benefits
  - Liabilities
- **Examples**
- **References**
  - Known Uses
## Example 1: Software Design Patterns

### Concurrency Patterns

#### Architectural Patterns
- Asynchronous Agents
- Parallel Tasks
- Repository
- Irregular Mesh

#### Algorithm Patterns
- Divide & Conquer
- Parallel Pipes & Filters
- Geometric Decomposition
- Recursive Data

#### Concurrency Patterns
- Half-Sync/Half-Async
- Leader/Followers
- Active Object
- Monitor Object
- Thread Specific Storage

#### Data Sharing Patterns
- Shared Data
- Shared Queue
- Replicable

#### Synchronization Patterns
- Thread-Safe Interface
- Double-Checked Locking
- Strategized Locking
- Scoped Locking

#### Program Structuring Patterns
- SPMD
- Master/Worker
- Loop Parallelism
- Fork/Join

#### Event Handling Patterns
- Proactor
- Reactor

---

### Classic concurrency patterns
- Origin: server applications
  - Many users
  - Small tasks that are more or less independent

### Parallel algorithms
- Scientific computations, high performance computing
- Image processing (data parallel algorithms)
- ...

---

### Multicore aspects not addressed
- Scalability with number of cores
- Memory hierarchy
- Parallel programming models

### Patterns Missing
- Only a few best practices are documented as design patterns; missing for example:
  - Patterns for task parallelism,
  - Speculative execution on application level,
  - Effective parallel stream processing

---

*SIEMENS*

*POSA2 Mattson et al.*
Example 1: Software Design Patterns

Example: Stream Processing Pipeline

Pipeline:

Parallel pipeline boosts throughput (Parallel Pipes-and-Filters pattern):

- no load balancing
- does not scale
- synchronization overhead for data transfer
- bad locality (memory hierarchy, cache effects)

Example 1: Software Design Patterns

Example: Parallel Pipeline Stages

Improvement of load balancing and throughput:

Multiple instances of long-running stages
- scalability still limited
- order gets lost, reordering might be necessary

Inside stages, data parallelism may be used
- improved latency
Example 1: Software Design Patterns

Example: Multiple Sequential Pipelines

Multiple sequential pipelines in parallel:

- good load balancing
- good cache locality
- but, order gets lost, reordering necessary

Example 1: Software Design Patterns

Conclusion

- Find a scalable partitioning of the problem
  - Architecture should support load balancing
  - Parallelism should be scalable with number of cores
  - Avoid waiting times

- Keep data local
  - Bad locality can slow down an application massively (costs for data transfer, false sharing)
  - No complicated architecture needed

- Parallel execution can change the processing order
  - Only possible if no dependencies between data elements
  - Additional effort for restoring order
Example 2: Consolidation

Real-Time Applications on Multicore Architectures

Industrial Control Units

- x86: HMI / Customer application
- ASIC: Real-time control (ARM)

Goals and work packages

- Integration of the real-time control (ASIC) on a multicore processor
- Cost savings (BoM, development), flexibility
- Design of an appropriate software architecture
- Experimental evaluation (load measurements)

Example 3: Acceleration Units

Application acceleration on GPUs (GPGPU)

Medical Image Processing

- Parallel CT reconstruction algorithms
- ECG segmentation and classification
- Ultrasound segmentation
- 3D Surface Generation
- Patient position tracking

Image Compression

- Accelerated discrete wavelet transform (APDCM workshop IPDPS '10)

Surveillance

- Real-time Face Processing
- presented @ HiPC '09
The Multicore Association (www.multicore-association.org)

Support for Complete System Design
- Improve time to market for applications through the use of standards
- MCA foundation APIs provide infrastructure to support other multicore services and value-added functions
- Communications and Resource Management are now in place; MTAPI will complete the foundation APIs to enable end-to-end multicore system development

MCA Foundation APIs

Communications (MCAPI)
- Lightweight messaging

Resource Management (MRAPI)
- Basic synchronization
- Shared/Distributed Memory
- System Metadata

Task Management (MTAPI)
- Task lifecycle
- Task placement
- Task synchronization

MTAPI Working Group – Multicore Association

Multicore Association Board Members

Multicore Association Working Group Members

Multicore Association University Members
Example 4: MTAPI
Task Management Programming Models

**Tasks**
- task parallel programming

**Queues**
- ordered execution

**Flow Graphs**

**Heterogeneous Architectures supported**
- shared memory
- non-shared memory
- different ISA (instruction set architectures)

**Resource Constraints**
- low memory footprint
- predictable behavior
- optimized to HW architecture

**Portability**
- plain C-API
- aligned with MCAPI and MRAPI
- different ISA, OS, bare metal

**Modularity**
- support different scheduling strategies (depends on problem to be solved and on hardware architecture)

Thank You!
shameless surreptitious advertising