Invasive Computing – The Quest for Many-Core Efficiency and Predictability

• What is Resource-Aware (Invasive) Computing?
  • Uniquitousness of parallel computers
  • Challenges in MPSoC design and programming in the year 2020
  • Potentials of Invasive Computing

• Structure of the DFG Transregional Research Center 89

• A Potpourri of Results
  • Foundations of Invasive Computing [A]
  • The parallel programming language X10 [C]
  • Invasive Multi-Tile Architectures [B]
  • Increasing Efficiency in HPC [D]

• InvasIC 2.0: A Look into the Future…
  • Invasive Computing for Predictability of Performance, Security, and Fault-Tolerance
Ubiquitousness of parallel computers

Nvidia Fermi: 512 Cores

Sony Playstation 3, IBM Cell 9 Cores

Intel SCC: 48 cores
Tightly-Coupled Processor Arrays (TCPAs)

Source: Hardware/Software Co-Design, Univ. of Erlangen-Nuremberg, Programmable 6x4 core TCPA MPSoC for image filtering. Technology: CMOS 1.0 V, 9 metal Layers 90 nm standard cell design. VLIW memory/PE: 16x128, FUs/PE: 2xAdd, 2xMul, 1xShift, 1xDPU. Registers/PE: 15. Register file/PE: 11 read/12 write ports. Configuration Memory: 1024x32 = 4 KB. Operating frequency: 200 MHz. Peak Performance: 24 GOPS. Power consumption: 132.7 mW @ 200 MHz (hybrid clock gating). Power efficiency: 0.6 mW/MHz.
Challenges in the year 2020

Architectures, Programming and Management of Applications for 1000s of Processors in 2020?

Challenges MPSoCs in Embedded Systems

• Complexity
  • How to map dynamically applications onto 1000 or more processors while considering memory, communication and computing resource constraints?

• Adaptivity
  • How and to what degree shall algorithms and architectures be adaptable (HW/SW, bit/word/loop/thread/process-level) and on behalf of which influencing factors (endogeneous (environmental) and indogeneous, e.g., availability and state of resources)?

• Predictability
  • How can QoS requirements such as on \{performance, safety, security, reliability,\ldots\} be guaranteed?
Invasive Computing

Diagram showing a system with multiple CPUs (CPU0, CPU1, CPU2, CPU3, CPU4), memory modules (MEM), and input/output devices (I/O). The CPUs are connected through a bus, with memory modules linked to the bus and CPU0 also connected to an I/O module. The diagram illustrates the flow of data and connections within the system.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Teich/Snelting</td>
<td>Henkel/Hübner/Bauer</td>
<td>Schröder-Preikschat/Lohmann/Henkel/Bauer</td>
<td>Dillmann/Asfour/Stechele</td>
</tr>
<tr>
<td>Sanders</td>
<td>Teich</td>
<td>Hannig/Gerndt/Herkersdorf</td>
<td>Bungartz/Gerndt</td>
</tr>
<tr>
<td></td>
<td>B3: Invasive Loosely-Coupled MPSoC</td>
<td>C3: Compilation and Code Generation for Invasive Programs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Herkersdorf/Henkel</td>
<td>Snelting/Teich</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B4: Hardware Monitoring System and Design Optimization for Invasive Architectures</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schmitt-Landsiedel/Schlichtmann</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B5: Invasive NoCs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Becker/Herkersdorf/Teich</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TRR 89 – Project Structure

Project Area A
Fundamentals, Language and Algorithm Design

Project Area B
Architectural Research

- B1
- B2
- B3
- B4
- B5

Project Area C
Compiler, Simulation, and Runtime Support

- C1
- C2
- C3

Project Area D
Applications

- D1
- D3

ALU
tightly-coupled
loosely-coupled
HPC
Outline

• What is Resource-Aware (Invasive) Computing?
  • Uniquitousness of parallel computers
  • Challenges in MPSoC design and programming in the year 2020
  • Potentials of Invasive Computing

• Structure of the DFG Transregional Research Center 89

• Potpourri of Results
  • Foundations of Invasive Computing [A]
  • The parallel programming language X10 [C]
  • Invasive Multi-Tile Architectures [B]
  • Increasing Efficiency in HPC [D]

• InvasIC 2.0: A Look into the Future…
  • Invasive Computing for Predictability of Performance, Security, and Fault-Tolerance
Basic Invasive Programming Constructs

• **In invade**
  – Allocation and reservation of system resources
    • Processors
    • Communication channels
    • Memory
  – Returns a claim (allocated resources)
  – Depends on the applications demand of parallelism
  – Depends on the current state of the resources (resource-aware)

• **Infect**
  • Copying program code and data to the claimed resources
  • Parallel execution of the program i-lets (code + data)

• **Retreat**
  • Frees occupied resources

Basics of Invasive Programming

-.permission
- speed
- utilization
- power/ temp
- fault/error
- invade
- infect
- retreat
Example of an Invasive MPSoC

[Diagram of an MPSoC with various components such as CPUs, NoC routers, memory, and iCore.]

**X10 Programming Language**

- **X10 Programming Language**
  - Parallel, object-oriented programming language
  - Developed by IBM (since 2004)

- **General Properties**
  - Supports distributed, heterogeneous processor and memory architectures
  - Syntax between Java and Scala
  - OO language features:
    - Classes, objects, inheritance, generic types
  - Functional language features:
    - Type inference, anonymous functions, closures, pattern matching
  - Parallel constructs:
    - Concurrency, synchronization, distribution, atomicity
  - PGAS Programming Model
PGAS Programming Model

• PGAS: Partitioned Global Address Space
  • Threads of a program have a global view, they share the same address space
    – Each thread sees the entire data set
    – No need for replication of data, as in the case of message passing
  • Address space is divided into partitions
    – Partitions may be physically distributed
    – Threads may reference data at other partitions (remote references)
    – Programmer is aware of data sharing among threads
X10 Parallel Constructs

- PGAS memory is called **Place** in X10
- PGAS thread is called **Activity** in X10

**Activity**
- Light-weight thread (user-level, not POSIX)
- Creation with `async`
- Synchronization via `finish`, `atomic`
- Activities cannot be named or aborted

**Place**
- Notion of a shared memory multi-processor
- Potentially different compute capabilities
- Holds activities and objects
- New places cannot be created at runtime

`async {S}`
- Creates a new child activity at the current place and asynchronously executes S
- Returns immediately

`finish {S}`
- Executes S and waits until all recursively spawned activities are finished

`at (P) {S}`
- Executes S at place P
- Current activity blocks
- Copy semantics
Case Study: Temperature-Aware Load Balancing

- Four PE tile
- Three job batch processing application
  - Allocating two PEs
- Maximum Temperature Constraint: 70°C

States of the application:
- idle
- invading
- infecting
- executing
- retreating

States of the PEs:
- idle
- invaded
- infected
- running

• **Basic control flow**

```scala
val claim = Claim.invade(constraints);
claim.infect(ilet);
claim.retreat();
```

• **Constraints**

```scala
val constraints = new AND();
constraints.add(new PEQuantity(2));
constraints.add(new MaximumTemperature(70));
constraints.add(new PlaceCoherence);
```

• **i-lets**

```scala
val ilet = (id:IncarnationID) => {
    Console.OUT.println("Hello from ilet " + id);
};
```
// main loop: a distributed image processing application
while (!terminated) {
val rnd = new Random();
val workload = new Array[Int]((0..1023)*(0..1023)),(Point)=>rnd.nextInt(255));
// invade
val claim = Claim.invade CONSTRAINTS);
// distribute workload to invaded PEs
val dist = PEDist.makeBlock(workload.region, claim);
val distWorkload = DistArray[int](dist, (p:Point) => workload(p));
val ilet = (id:IncarnationID) =>
    { // ... code to execute on each claimed PE
        // return a reference to the local result
        new RemoteArray[int](localResult) );
    // infect
val results = claim.infect[RemoteArray[int]](ilet);
val result = new Array[int] (nbins);
val tmp = new Array[int] (nbins);
for ([i] in results) {
    finish Array.asyncCopy(results(i),tmp);
    for ([j] in tmp)
        result(i) += tmp(i);
}
// retreat
claim.retract();
}
Constraint Hierarchy

SetConstraints
  \- PEQuantity
  \- PartitionConstraints
    \- PlaceCoherence
    \- TypeHomogeneity
    \- CacheHomogeneity

OrderConstraints
  \- OrderedByLoad
  \- OrderedByTemperature
  \- OrderedByMemory
  \- OrderedBySpeed

PredicateConstraints
  \- MaximumLoad
  \- MaximumTemperature
  \- FpuAvailable
  \- LocalMemory
  \- ScratchPadSize
  \- TypeConstraint
  \- CacheConstraint
  \- Migratable
  \- NonPreemptible
  \- TopoLayout

MultipleConstraints
  \- AND
  \- OR

Hint
  \- EfficiencyCurve

Invasive TCPAs

• Each PE is equipped with an invasion controller (i-Ctrl) to enable a cycle-wise, decentralized resource reservation [ASAP11]

• Invasion support:
  – Different resource exploration methods (invasion strategies) [IPDPSW11]
    • Linearly connected region of PEs [VLSISoC09]
    • Rectangular connected region of PEs
  – Different designs for invasion controller:
    • FSM-based [FPT2009]
    • Programmable [ASAP 2011]
TCPA Invasion for Power Management


- Retreat phase works similar

PMU: Power Management Unit
Increasing Efficiency in HPC

Invasive tsunami simulation on dynamic adaptive grids

Invasive resource manager accounts for application’s provided resource requirements

Invasive Computing

- Consideration of changing resource requirements.
- Start applications despite insufficient resources available.

=> Improved total efficiency and performance!
Outline

• What is Resource-Aware (Invasive) Computing?
  • Uniquitousness of parallel computers
  • Challenges in MPSoC design and programming in the year 2020
  • Potentials of Invasive Computing

• Structure of the DFG Transregional Research Center 89

• A Potpourri of Results
  • Foundations of Invasive Computing [A]
  • The parallel programming language X10 [C]
  • Invasive Multi-Tile Architectures [B]
  • Increasing Efficiency in HPC [D]

• InvasIC 2.0: A Look into the Future…
  • Invasive Computing for Predictability of Performance, Security, and Fault-Tolerance
1. Observation: Many computationally-intensive embedded system applications such as audio and video (image) processing do not opt for maximizing performance but to **guarantee a throughput requirement**.
   Example: Invasive video filter demonstration
   - Applications: FIR filter (1D), edge detection (2D)

2. Observation: Many other embedded applications **require a maximal reaction time**, resp. processing latency
   Example:
   - Image Analysis for collision detection/prediction
Why tight QoS predictability on TCPAs?

- Once claimed, a processor region is **static** and **exclusive**
- Architecture Design **prevents interference** during i-let execution through avoidance of
  - cache eviction
  - dynamic thread scheduling
  - interrupts
  - OS overheads
  - other i-let executions
- **Symbolic Loop Scheduling** allows to determine **exact throughput and latency bounds** depending only on claim size
Example: TCPAs for QoS Predictability

- Sobel 1x3
- Laplace 3x3
- Laplace 5x5

**Throughput**

- **Claim size:** #PES
- **Throughput:**
  - 1x3
  - 3x3
  - 5x5

**Quality**

- 0 1 2 3 4
Real-time video processing on invasive TCPA
Timing Predictability Over Multiple Tiles

- **Camera I/O**
- **Edge Detection**
- **Feature Extraction**
- **Video I/O**

### Invasive HW/SW Blocks

<table>
<thead>
<tr>
<th>Applications</th>
<th>Leon3</th>
<th>iCore</th>
<th>TCPA</th>
<th>Data throughput (iNoC)</th>
<th>ilet injection rate *(CIC)</th>
<th>OctoPOS, Agents, Monitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optical Flow</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>Used</td>
</tr>
<tr>
<td>Harris Corner Detection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>Used</td>
</tr>
<tr>
<td>SIFT Feature Extraction</td>
<td>Yes</td>
<td>No</td>
<td>Low</td>
<td>High</td>
<td>Med</td>
<td>Used</td>
</tr>
<tr>
<td>Feature Matching</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>High</td>
<td>Low/med</td>
<td>Used</td>
</tr>
<tr>
<td>Disparity Map</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>High</td>
<td>Low</td>
<td>Used</td>
</tr>
<tr>
<td>Motion Planning</td>
<td>Yes</td>
<td>-</td>
<td>No</td>
<td>Low</td>
<td>Med</td>
<td>Used</td>
</tr>
<tr>
<td>Audio</td>
<td>Yes</td>
<td>-</td>
<td>No</td>
<td>Medium</td>
<td>High</td>
<td>Used</td>
</tr>
</tbody>
</table>
Exclusiveness through Invasion -> Predictability
Exclusiveness of Invasion => Predictability

\[ T_{m1} + T_{ED} + T_{m2} + T_{FE} + T_{m3} + T_M \leq T_{limit} \]
Enabling Security in/of Invasive Computing Systems

Virtual InvasIC A

Virtual InvasIC B

Virtual InvasIC C
• Introduction of a new paradigm of resource-aware programming as well as new architectural support by reconfigurable MPSoC architectures: InvasICs

• Expected impact on:
  Future advanced processor development for MPSoCs
  Future programming environments for many-core systems
  Development of parallel algorithms
  *-Predictability

• Potential Risks:
  Acceptance of resource-aware programming
• DFG Transregional Collaborative Research Center 89 – Invasive Computing, [www.invasic.de](http://www.invasic.de)